Advant® OCS
with Master software

GCOM™ Multidrop

User's Guide

Use of **DANGER**, **WARNING**, **CAUTION**, and **NOTE**

This publication includes, **DANGER**, **WARNING**, **CAUTION**, and **NOTE** information where appropriate to point out safety related or other important information.

DANGER	Hazards which could result in severe personal injury or death
WARNING	Hazards which could result in personal injury
CAUTION	Hazards which could result in equipment or property damage
NOTE	Alerts user to pertinent facts and conditions

Although **DANGER** and **WARNING** hazards are related to personal injury, and **CAUTION** hazards are associated with equipment or property damage, it should be understood that operation of damaged equipment could, under certain operational conditions, result in degraded process performance leading to personal injury or death. Therefore, comply fully with all **DANGER**, **WARNING**, and **CAUTION** notices.

TRADEMARKS

Master, MasterPiece and MasterView are registered trademarks of ABB Asea Brown Boveri Ltd. Schweiz.

MasterBus is a trademark of ABB Asea Brown Boveri Ltd. Schweiz.

Advant is a registered trademark of ABB Process Automation Corp. USA.

NOTICE

The information in this document is subject to change without notice and should not be construed as a commitment by ABB Industrial Systems AB. ABB Industrial Systems AB assumes no responsibility for any errors that may appear in this document.

In no event shall ABB Industrial Systems AB be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall ABB Industrial Systems AB be liable for incidental or consequential damages arising from use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without ABB Industrial Systems AB’s written permission, and the contents thereof must not be imparted to a third party nor be used for any unauthorized purpose.

The software described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license.

Copyright © ABB Industrial Systems AB 1997

3BSE001264/C

TABLE OF CONTENTS

Chapter 1 - Introduction

1.1 General Information 1-1
1.2 Equipment Requirements 1-1
1.3 Manual Organization 1-1
1.4 Conventions 1-3
1.5 Related Documentation 1-3
1.6 Release History 1-4
1.7 Terminology 1-4
1.8 Product Overview 1-5
  1.8.1 GCOM Characteristics 1-5
  1.8.2 GCOM/IEEE 802.3 Format with Protocol Identifier 1-7
  1.8.3 Address Handling 1-8
  1.8.4 Transit Handling 1-9
  1.8.5 Data Transfer Handling 1-10
  1.8.6 Acknowledge and Sequence Handling 1-12
1.9 User Interface 1-13

Chapter 2 - Installation

2.1 Site Planning Environment 2-1
2.2 Setup 2-1
2.3 Shut-down Procedures 2-1
2.4 Start-up Procedures 2-1
2.5 Product Verification 2-1

Chapter 3 - Configuration/Application Building

3.1 Design Considerations 3-1
  3.1.1 Object Functions 3-1
  3.1.2 Application Functions 3-1
  3.1.3 System Functions 3-1
3.2 Capacity and Performance 3-2
3.3 Application Start-up 3-2
3.4 Tutorial 3-3
  3.4.1 “I am here” message 3-5
  3.4.2 Communication Status 3-6
  3.4.3 Dataset Request 3-8
  3.4.4 Symbolic Name Translation 3-10
  3.4.5 A Data Subscription Request (demand) 3-12

CONTENTS (continued)

3.5 Application Procedures 3-13
  3.5.1 Application Layer........ 3-14
  3.5.2 Address Part 3-15
    3.5.2.1 Messages to ABB Master System. 3-16
    3.5.2.2 Messages to the External Computer........ 3-17
  3.5.3 Data Part Object Functions 3-18
    3.5.3.1 Data Subscription 3-20
    3.5.3.2 Answer on Data Subscription........ 3-24
    3.5.3.3 Historical Time Tagged Data (TTD)........ 3-26
    3.5.3.4 Order........ 3-39
    3.5.3.5 System Text........ 3-44
  3.5.4 Data Part Application Function........ 3-45
    3.5.4.1 Data Set Communication (DSC) 3-45
  3.5.5 Data Part System Functions 3-46
    3.5.5.1 Symbolic Name Translation........ 3-47
    3.5.5.2 Clock Synchronization 3-49
    3.5.5.3 Communication Status 3-50
    3.5.5.4 Event Handling........ 3-52
3.6 Configuration/Application Building Menus 3-55
  3.6.1 Data Subscription........ 3-55
    3.6.1.1 Pascal Layout Subscription Request and Clear........ 3-55
    3.6.1.2 Pascal Layout mmi clear 3-55
  3.6.2 Answer on Data Subscription........ 3-56
    3.6.2.1 Pascal Layout for AI and AO Objects........ 3-56
    3.6.2.2 Status for AI and AO Objects........ 3-60
    3.6.2.3 Pascal Layout for DI and DO Objects........ 3-62
    3.6.2.4 Status for DI and DO Objects........ 3-63
    3.6.2.5 Pascal Layout for PIDCON, RATIO and MANSTN..... 3-65
    3.6.2.6 Parameters and Status for PIDCON, RATIO and MANSTN3-76
  3.6.3 Historical Time Tagged Data (TTD)........ 3-88
    3.6.3.1 Pascal Layout for Subscription........ 3-88
    3.6.3.2 Pascal Layout for Data Answer........ 3-90
    3.6.3.3 Pascal Layout Log Variable 3-91
  3.6.4 Order 3-93
    3.6.4.1 Pascal Layout Order 3-93
    3.6.4.2 Pascal Layout Select Answer 3-94

CONTENTS (continued)

3.6.5 System Text 3-94
3.6.5.1 Pascal Layout syst_text 3-94
3.6.5.2 Text Pointers 3-95
3.6.6 Communication Status 3-99
3.6.6.1 Pascal Layout Communication Status 3-99
3.6.7 Event Handling 3-100
3.6.7.1 Pascal Layout System Events 3-100

Chapter 4 - Runtime Operation

4.1 Product Operation 4-1
4.2 Operating Overview 4-1
4.3 Runtime Tutorial 4-1
4.4 Operating Instructions 4-1
4.5 Runtime Operation Menus 4-1

Chapter 5 - Maintenance

5.1 Preventive Maintenance 5-1
5.2 Hardware Indicators 5-1
5.3 Error Messages 5-1
5.4 Fault Finding and User Repair 5-1

Appendix A - Data Representation

A.1 Internal Representation A-1
A.1.1 Integer A-1
A.1.2 Character A-2
A.1.3 Boolean A-2
A.1.4 Set A-2
A.1.5 Enumerated scalar A-3
A.1.6 String A-3
A.1.7 Pointer A-3
A.1.8 Real A-4
A.2 Data Representation in the GCOM Protocol A-4
A.3 Definitions A-5

Appendix B - Global Channel Identities

B.1 Gloabal Channel Identities B-1

Appendix C - Signal Identities

C.1 Signal Identities C-1

CONTENTS (continued)

Appendix D - Operation Codes and Properties

D.1 Operation Codes...D-1
D.2 Operation Properties...D-2

INDEX

Chapter 1 Introduction

1.1 General Information

This manual provides you with the information you need to use the GCOM protocol on a multidrop bus. It is not designed to provide detailed information on how, for example, a PIDCON object works. It is intended for programmers and users of the GCOM protocol in external computers.

The GCOM protocol is implemented as a data link user. It is used by external computers to gain access to the ABB Master task-to-task communication service.

GCOM subnet is the same as DSX/IEEE 802 subnet.

GCOM Multidrop provides a powerful and easy-to-implement communication link from external computers to the ABB Master system.

The GCOM protocol makes external computers members of the ABB Master communication system (MasterNet). It enables them to use almost all of the powerful functions in the ABB Master system.

1.2 Equipment Requirements

For equipment requirements, see the *MasterNet User's Guide* or *MasterGate Installation and Error Codes*, depending on your configuration.

1.3 Manual Organization

Figure 1-1 shows the basic structure of Advant documentation. Each document, describing hardware or software, is built around this one structure. Using one structure makes it easy for you to locate related information in any of the documents. If a certain section, or an entire chapter, is not relevant to a particular Advant System document, the section is still included (with a note indicating that it is not relevant) in order to remain consistent with the structure.

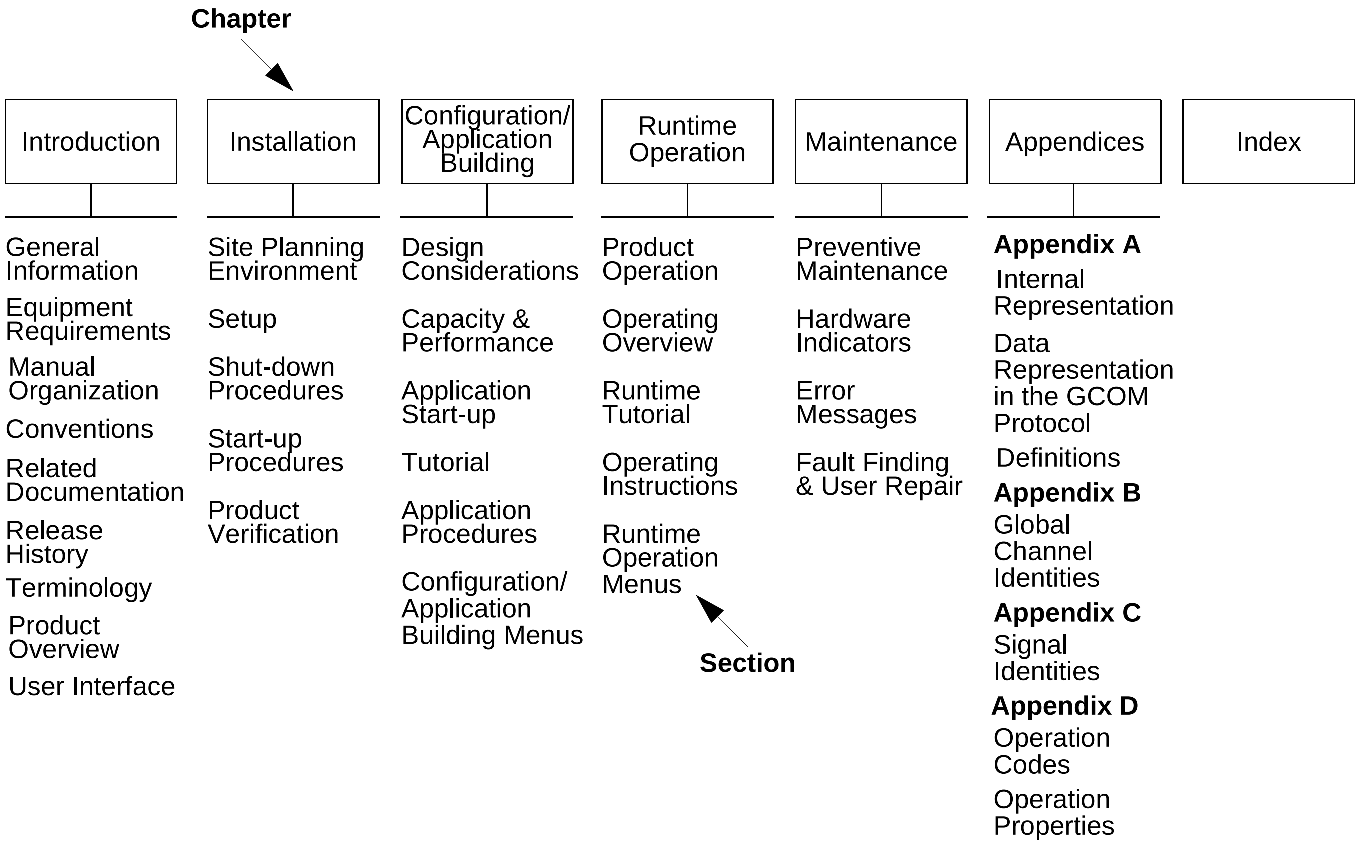


Figure 1-1. Manual Organization Diagram

Introduction

The Introduction gives you basic information about this manual and GCOM Multidrop.

Installation

Not applicable. Installation of GCOM in ABB Master products is described in the manual for the corresponding product. Installation of GCOM in the ABB Master Advant Controller 450 is described in the *Master Net User's Guide*.

Configuration and Application Building

Chapter 3 describes the **dsx** (Distributed Signal Exchange) signals, both graphic and Pascal layout, of the address and data parts.

Runtime operation

Not applicable.

Maintenance

Not applicable. Maintenance of GCOM in ABB Master products is described in the manual for the corresponding product. Maintenance of GCOM in the ABB Master Advant Controller 450 is described in the *Master Net User's Guide*.

1.4 Conventions

All values are in decimal format if not stated otherwise.

1.5 Related Documentation

Refer to the following documents for further information regarding ABB Master products with GCOM communication.

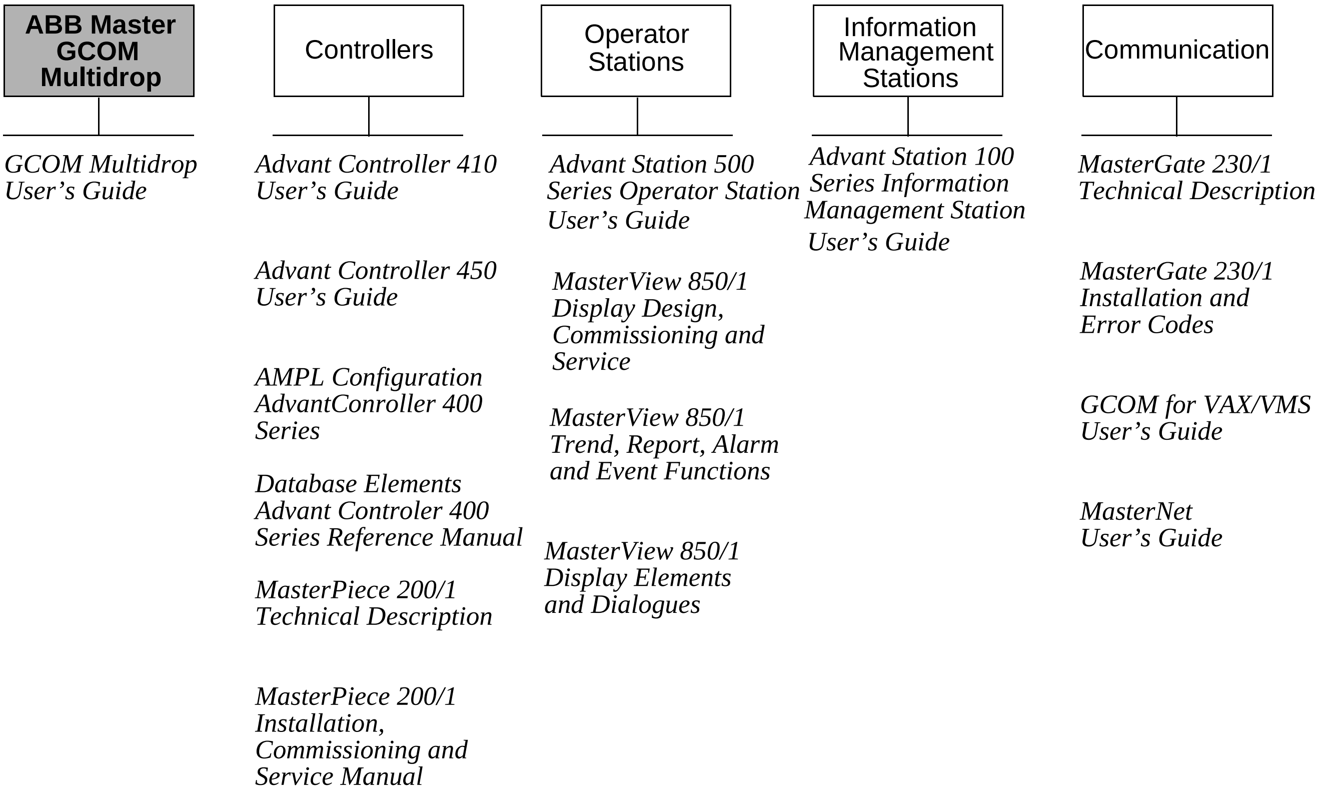


ABB Master GCOM Multidrop
GCOM Multidrop User's Guide
Controllers
Advant Controller 410 User's Guide
Advant Controller 450 User's Guide
AMPL Configuration AdvantConroller 400 Series
Database Elements Advant Controler 400 Series Reference Manual
MasterPiece 200/1 Technical Description
MasterPiece 200/1 Installation, Commissioning and Service Manual
Operator Stations
Advant Station 500 Series Operator Station User's Guide
MasterView 850/1 Display Design, Commissioning and Service
MasterView 850/1 Trend, Report, Alarm and Event Functions
MasterView 850/1 Display Elements and Dialogues
Information Management Stations
Advant Station 100 Series Information Management Station User's Guide
Communication
MasterGate 230/1 Technical Description
MasterGate 230/1 Installation and Error Codes
GCOM for VAX/VMS User's Guide
MasterNet User's Guide

Figure 1-2. GCOM Multidrop and Related Documentation Tree

1.6 Release History

Major milestones in the product release are:

Table 1-1. GCOM Multidrop Release History

Version	Description
DSCS 150*1.0	First version of GCOM for MG 230/1*3.0 or later and MP 200/1*2.1 or later
DSCS 150*1.1	GCOM with variable frame size.
SC530*3.0	First version for AC 450*1.0

1.7 Terminology

- **AC** is an abbreviation for Advant Controller.
- **AI** is an Analog Input signal.
- **AC 400** is an abbreviation for the ABB Master Advant Controller 400 Series.
- **AMPL** is a function-block language with graphic presentation which is especially oriented towards process control. The AMPL is used for configuration and application building in the ABB Master Controllers.
- **AO** is an Analog Output signal.
- **Controller** refers to ABB Process Stations and Master Controllers (MP 200/1, AC 400, AC 110, etc.).
- **Cyclic subscription** is a standing subscription to data by cyclic reporting.
- **Demand subscription** is data collection on random demand.
- **DAT** is a concept in the database that holds a data value (1 Real- (4 byte), 1 Integer- (2 or 4 byte) or 32 Boolean values). The data values to DS are held by connected DAT elements.
- **DI** is a Digital Input signal.
- **DO** is a Digital Output signal.
- **DS** is an abbreviation for Data Set, which is a data base concept in ABB Master. It is used to transfer blocks of data in an ABB Master network.
- **DSX** is an abbreviation for Distributed Signal Exchange.
- **Event subscription** is a standing subscription to data by event-driven reporting.
- **LLC** is an abbreviation for Logical Link Control.
- **MAC** is an abbreviation for Medium Access Control.

- **MB 300/300E** are high-speed buses used to interconnect Controllers, Operator Stations and Communication Stations.
- **MP** is an abbreviation for the ABB MasterPiece process station.
- **PDU** is an abbreviation for Protocol Data Unit.
- **TTD** is a log with time-tagged historical data.

1.8 Product Overview

1.8.1 GCOM Characteristics

The GCOM Multidrop bus (GCOM subnet) is a high-performance serial-synchronous half-duplex bus for medium communication distances.

The GCOM's simple three-layer structure, multidrop bus connection and freedom from segmentation make it easy to implement in an external computer, see Figure 1-3.

The protocol is implemented as a datalink user, operating on a standard IEEE 802.3 bus. The multidrop bus has an IEEE 802.2 class 1 connectionless unconfirmed data link service. The bus has no specific master station, i.e., all stations have equal access to the bus.

The GCOM subnet has high security through the flow control and retransmission handling in the GCOM-layer.

Broadcast is not supported on the GCOM subnet.

On the ABB Master side, the GCOM communication software is implemented on a microprocessor-based communication board (DSCS 150 in MG 230/1 and MP 200/1, SC530 in AC 450).

The GCOM subnet (bus) can handle up to four (4) external computers on the same bus and the transmission rate can be up to 10 Mbit/s.

The GCOM protocol is also implemented by ABB as an IEEE 802.3 datalink user on VAX computers.

Application Tasks	Application
GCOM Flow Control	Transport
LLC IEEE 802.3	Data Link
MAC IEEE 802.3	
PHY IEEE 802.3	Physical

Figure 1-3. GCOM Structure

As you can see in the table above, there is a “null” network layer and a minimal transport layer.

The GCOM communication subnet allows user tasks to send messages between stations, connected to the same IEEE 802.3 bus within an ABB Master Control Network or Plant Network.

With the transit function in MasterNet, a user task on the GCOM subnet can send and receive signals to/from any ABB Master station in a MasterNet Control Network or Plant Network.

Figure 1-4 shows an example of a configuration including an external computer.

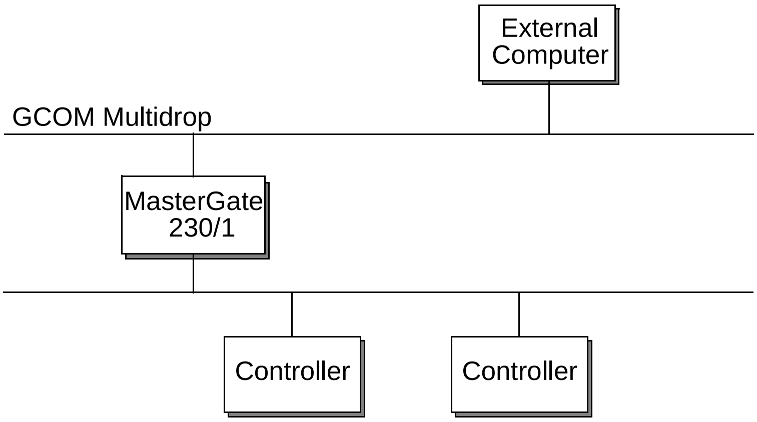


Figure 1-4. Example of an configuration

1.8.2 GCOM/IEEE 802.3 Format with Protocol Identifier

The structure of a GCOM message is shown in Figure 1-5.

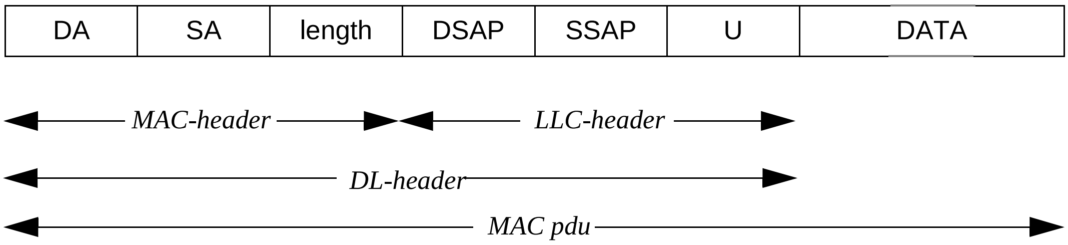


Figure 1-5. MAC pdu

MAC pdu:

- DL-header
  - MAC-header

    DA: Destination address (6 bytes)

    SA: Source address (6 bytes)

    Length: Length of DSAP, SSAP, U and DATA.
  - LLC-header

    DSAP: Destination service access point (1 byte)

    SSAP: Source service access point (1 byte)

    U: Unnumbered control field command/response. Always **3** (1 byte).
- Data part

  DATA: user-supplied data plus padding (PAD)

The dl-header is normally handled by the DL-layer.

NOTE

The MAC pdu must always have a minimum number of bytes. If the MAC pdu is less than the minimum number of bytes, it must be padded to the minimum number of bytes (510 for fixed frames or 64 for variable frames).

For specifications on how to set up the MAC frame size, see *MasterGate Installation and Error Codes* (for MasterGate) and the *MasterNet User’s Guide* (for Advant Controller 450 and MasterPiece 200/1).

1.8.3 Address Handling

Each station in the total ABB Master system is identified with a logical address (network and node number). The station addresses (DA and SA) are the physical addresses on the GCOM subnet to which the external computer is connected. ABB Master units on an IEEE 802.3 bus use the following station address (hexadecimal value):

SA: **< 00, 00, 23, 00, nn, 01 >**

where nn = the logical node number for the ABB Master unit to which the GCOM subnet is physically connected.

The source address for the external computer is normally defined by its manufacturer.

The logical address in the data part (network and node number) defines the logical address for a unit in the MasterNet network.

For information about the multicast address, see Section 1.8.4.

1.8.4 Transit Handling

Transit handling is used to build up a cross-reference table between station addresses on the GCOM subnet and logical addresses (network and node number) in the ABB Master system.

Both the external computer and the ABB Master unit on the GCOM subnet cyclically (every one second) send an “I am here” message with the MasterNet-GCOM multicast MAC address. A station is not active on the GCOM subnet if the other stations have not received a new “I am here” message within three seconds.

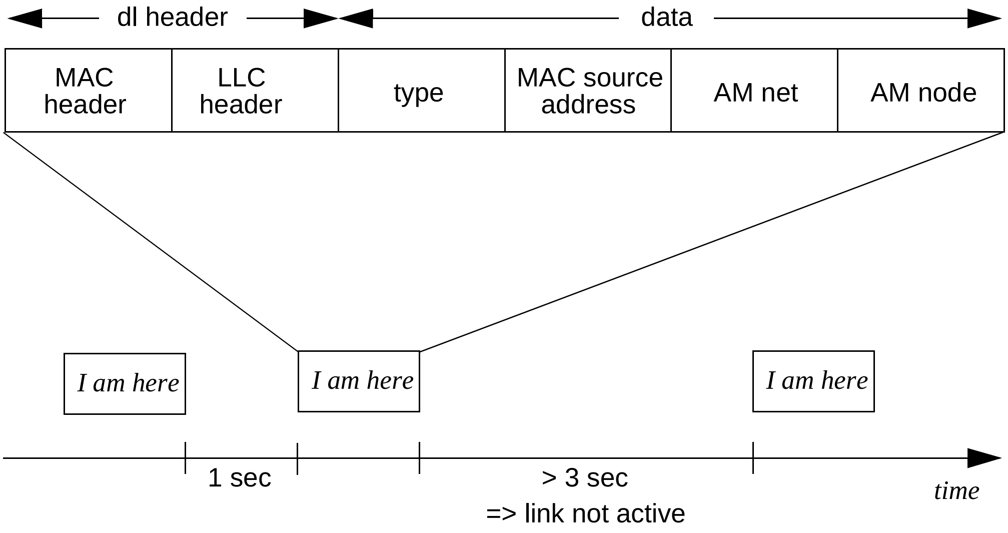


Figure 1-6. “I am here” message

- DL header
  - **MAC**: The destination address (DA) is MasterNet-GCOM multicast address (hexadecimal value) **< 01, 00, 23, 00, 00, 01 >**.
  - **LLC**: DSAP and SSAP (hexadecimal value) = **< 10 >**.
- Data part
  - **Type**: 1 byte where bit 0-3 defines the type of pdu and bit 4-7 is not used. For “I am here” pdu, TYPE = **4**.
  - **MAC source address**: 6 bytes which identify the IEEE 802 MAC address of the source station. This 6-byte address is transmitted as defined by IEEE for the MAC header.
  - **AM net:** 1 byte which identifies the ABB Master subnet to which the alien computer is connected. AM net = **11-99; 110-119**.
  - **AM node:** 1 byte which identifies the source station within the ABB Master DSX/IEEE 802 subnet. For ABB Master stations, AM node = **1-99, 127**; for alien computers, AM node = **10-99**.

1.8.5 Data Transfer Handling

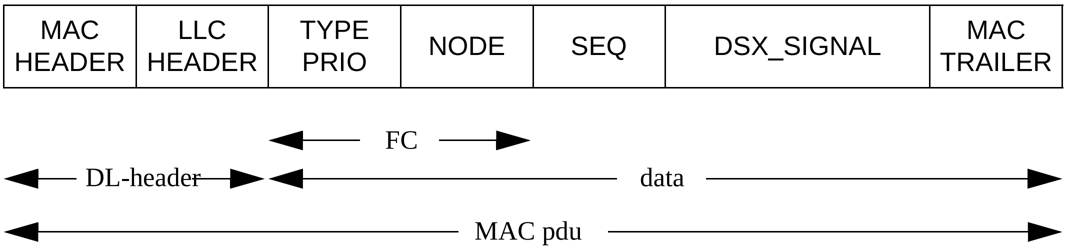


Figure 1-7. Data Transfer pdu

Data transfer pdu:

- DL header
  - **MAC:** See Section 1.8.2.
  - **LLC:** DSAP and SSAP (hexadecimal value) = **< 14 >**.
- Data part
  - **FC (**Flow Control Header)

    TYPE/PRIO: 1 byte, where bit 0-3 defines the type of pdu and bit 4-7 defines the dsx signal priority.

    Bit 0-3: TYPE = **0** for Data Transfer pdu

    Bit 4-7: value **2** = low priority
    value **3** = normal priority
    value **4** = high priority
    value **5** = emergency priority
    The external computer always uses normal priority in data transfer pdus to ABB Master.

    NODE: 1 byte which is the source node number, **10-99** for the alien computer, **1-99, 127** for ABB Master stations.
  - **SEQ:** 1 byte which defines the send sequence counter modulo 128.
  - **DSX_SIGNAL:** see Section 3.5.

Two (2) sequence counters are used, one (1) for signals to (S) and one (1) for signals from (R) ABB Master. A Data Transfer pdu (dtpdu) must be acknowledged within **250 ms**. If acknowledge is not received, the dtpdu should be retransmitted with the same sequence number. Each signal can, if necessary, be retransmitted up to two times.

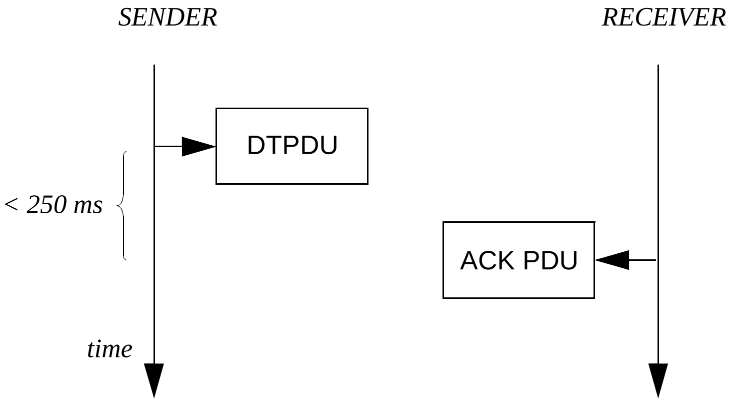


Figure 1-8. Acknowledge of a dtpdu

For more information, see Section 1.8.6.

1.8.6 Acknowledge and Sequence Handling

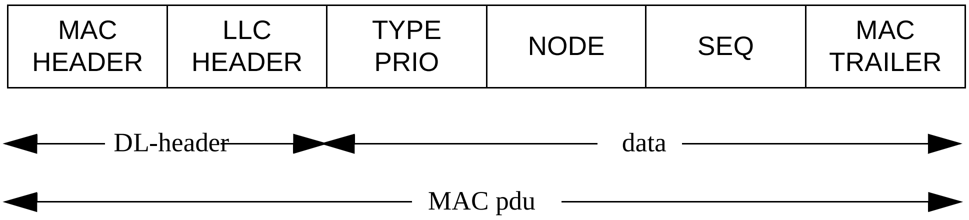


Figure 1-9. Acknowledge or Init pdu

Acknowledge or init pdu:

- DL-header
  - **MAC:** see Section 1.8.2.
  - **LLC:** DSAP and SSAP (hexadecimal value) = **< 14 >**.
- ACK and INIT data part
  - **TYPE/PRIO:** 1 byte, where bit 0-3 defines the type of pdu and bit 4-7 defines the dsx signal priority.

Bit 0-3:	TYPE = **1** for ACK pdu
	TYPE = **2** for INIT_R pdu
	TYPE = **3** for INIT_S pdu
Bit 4-7:	value **2** = low priority
	value **3** = normal priority
	value **4** = high priority
	value **5** = emergency priority

  - **NODE:** 1 byte which is the source node number. **10-99** for the alien computer, **1-99, 127** for ABB Master stations.
  - **SEQ:** 1 byte which defines the send sequence counter modulo 128. SEQ identifies the next segment expected to be received and that the previous segment has been successfully received. If TYPE=**INIT** then SEQ=**0**.

Two (2) sequence counters are used, one (1) for signals to (S) and one (1) for signals from (R) ABB Master.

A received signal from the ABB Master system can be one of four types:

1. If data (dtpdu) is received and no transmission or sequence error (SEQ=R) is detected, the signal is acknowledged with SEQ=R+1 (receive sequence counter is increased by one (1)). A dtpdu must be **acknowledged by the receiver within 250 ms.**

   If a dtpdu is received with SEQ=R-1 (and no transmission error is detected), it is a retransmission of data. The dtpdu must be acknowledged with SEQ=R (retransmission of ACK).

   If a dtpdu is received with sequence error (SEQ<R-1 or SEQ>R), an INIT_S message must be sent to the ABB Master unit. The sequence counter R must be reset (R=0) and the received data signal is dropped.

2. If an ACK is received with SEQ=S+1, it means that the ABB Master unit has received the previously sent message and gives credit to send the next signal, i.e., the signal has been successfully received by the ABB Master unit.

   If the send sequence counter (S) is out of sequence (SEQ<0, SEQ >127 or SEQ≠S+1), then the send sequence counter should be re-initialized to 0 and an INIT_R message must be sent to the ABB Master unit.

3. If an INIT_R message is received, it means that the receive sequence counter must be reset to 0.

4. If an INIT_S message is received, it means that the send sequence counter must be reset to 0.

1.9 User Interface

Not applicable.

Chapter 2 Installation

2.1 Site Planning Environment

You can install the GCOM multi-drop communication in either a MasterGate 230/1, an Advant Controller 400 Series or a MasterPiece 200/1 unit.

If you want to install GCOM in an AC400 or an MP 200/1, the GCOM subnet is a part of the Control network to which the AC400/MP 200/1 belongs. For an installation guide, see the *MasterNet User's Guide*.

If you want to install GCOM in an MG 230/1, the GCOM subnet is a part of the Plant network. For an installation guide, see the *MasterGate Installation and Error Codes*.

NOTE

ABB takes no responsibility for ill-behaved external computers effect on automation systems from ABB Industrial Systems.

2.2 Setup

Not applicable.

2.3 Shut-down Procedures

Not applicable.

2.4 Start-up Procedures

Not applicable.

2.5 Product Verification

Not applicable.

Chapter 3 Configuration/Application Building

3.1 Design Considerations

When the external computer uses the GCOM protocol, the application programmer can use powerful functions in the ABB Master system.

The external computer can use three types of functions in the ABB Master system:

- Object functions
- Application functions
- System functions.

3.1.1 Object Functions

Data subscription for objects. Data subscription is a function that collects one or several specified properties (parameters) from one or several specified database references (objects). A subscription request message may consist of several object types from the same ABB MasterPiece or Advant Controller. The collected properties are sent in different response messages, depending on object type (reference type).

Data subscription for Time Tagged Data (TTD). External computers may subscribe to historical Time Tagged Data (TTD) from the ABB MasterPieces or Advant Controllers.

Order. With the function order, the external computer can control and change parameters for objects in the ABB Master system. Objects which are continuously controlled by external computers should be handled by the DSC function.

3.1.2 Application Functions

Data Set Communication (DSC). The DSC function transfers data cyclically between the external computer and the ABB Master databases in the form of Data Set Packets (DSPs). External computers can also get DSPs from ABB Master on request.

3.1.3 System Functions

Symbolic name translation. The external computer can send the symbolic name of an object to a MasterPiece or Advant Controller. The MasterPiece or Advant Controller translates the name to a logical database reference (LF and LR (refadr)) and returns it to the external computer. With this function, the external computer can build up a cross-reference table between symbolic names and logical database addresses in the ABB Master system.

Clock synchronization. This function synchronizes the system clocks in both systems.
The ABB Master system or one of the external computers connected to the GCOM subnet can handle the clock synchronization.

Communication status. This function, communication status (system status), enables the external computer to supervise units (directly) connected to an ABB Master unit. The request must be made cyclically.

Event handling. The external computer can subscribe to system events from the ABB MasterGate (if it is connected to an MG 230).

The ABB Master Station units in the ABB Master system can use the same functions against the external computer (if the functions are implemented on the application layer with the same interface as in the ABB Master system).

3.2 Capacity and Performance

The performance of GCOM protocol depends on where it is installed. For example, if the GCOM communication board is installed in a MasterGate, the following is valid:

A maximum of 50 messages per second can pass one MasterGate 230/1 (valid when GCOM multidrop is used).

Each MasterPiece or Advant Controller can handle a maximum of five messages per second to/from its own MasterGate230/1.

With the function data subscription, an external computer can work against up to 300 objects at one time. Of those, a maximum of 30 can have a cycle time of three seconds; the remaining objects must have a cycle time of nine seconds or be of the demand type.

Examples of messages follow:

- One data subscription request with one to 16 objects.
- An answer on a request which can consist of one to 38 objects, depending on the object type and access type.
- One dataset, with up to 24 values (a value can be one real, one integer (2 or 4 bytes) or 32 digital).
- One order.

An answer on a data subscription request for historical time-tagged data (TTD) consists of up to seven messages.

If the external computer uses only the DSC function, it can send/receive 24x50 = 1200 values per second (1200 integer/real or 38400 boolean object values).

If the external computer uses only the function data subscription and all values are logged and stored as historical time tagged data in the ABB MasterPieces or Advant Controllers, then the external computer can request 402 values in one TTD request. The answer is received in seven messages. The external computer then receives up to (50/(7+1))402 = 2500 values per second.

With the function data subscription, an external computer can work against up to 300 objects at one time. Of those, a maximum of 30 can have a cycle time of three seconds. The remainder must have a cycle time of nine seconds or be of the demand type.

3.3 Application Start-up

Not applicable.

3.4 Tutorial

This tutorial gives an example of signals sent on the IEEE 802.3 link when data set and subscription of data are requested from an analog input from an Advant Controller. Programming examples can also be found in the *GCOM for VAX/VMS User's Guide*. This tutorial is based on the configuration below (see Figure 3-1).

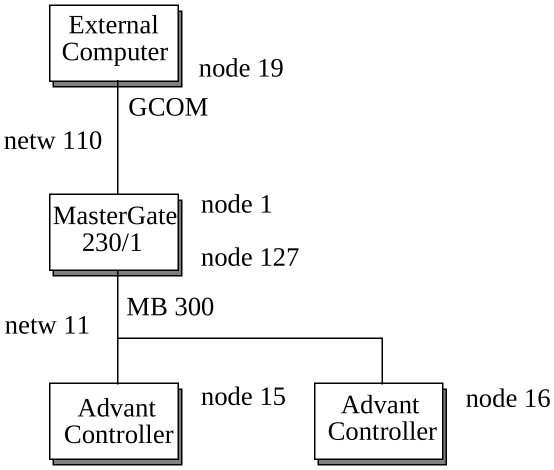


Figure 3-1. Tutorial Configuration

The following signals are described:

1. “I am here” messages from the external computer and from the MG 230/1.
2. A communication status request from the external computer to the MG 230/1 and the answer.
3. A data set request to netw 11 and node 15 and the answer from the AC.
4. A symbolic name translation of the object AIC16_1.
5. A data subscription of type demand on AIC16_1 (node 16).

The data transfer pdu sent from the external computer differ only in length and in the DSX signal part. The pdu appears as follows. (For further description of the signal, see Section 1.8.5 Data Transfer Handling.)

1	7	13	15	16	17
MAC destination address	MAC source address	length	DSAP	SSAP	U

18	19	20	21	
type prio	node	seq	DSX signal	MAC trailer

MAC destination address: **< 00, 00, 23, 00, 01, 01 >** (hexadecimal value)
MAC source address: **< 00, 00, 23, 00, 19, 01 >** (hexadecimal value)
Length: depends on the DSX signal
DSAP: **14** (hexadecimal value)
SSAP: **14** (hexadecimal value)
U: **3** (always)
Type/prio: bit 0-3: **0** (data transfer message)
bit 4-7: **3** (normal priority)
node: **19**
seq: **0-127**
DSX signal: see sections below for each type of data transfer signal
MAC trailer: padding and checksum

For every message sent, an acknowledge is sent back. The acknowledge pdu appears as follows. (For further description of the signal, see Section 1.8.6 Acknowledge and Sequence Handling.)

1	7	13	15	16	17
MAC destination address	MAC source address	length	DSAP	SSAP	U

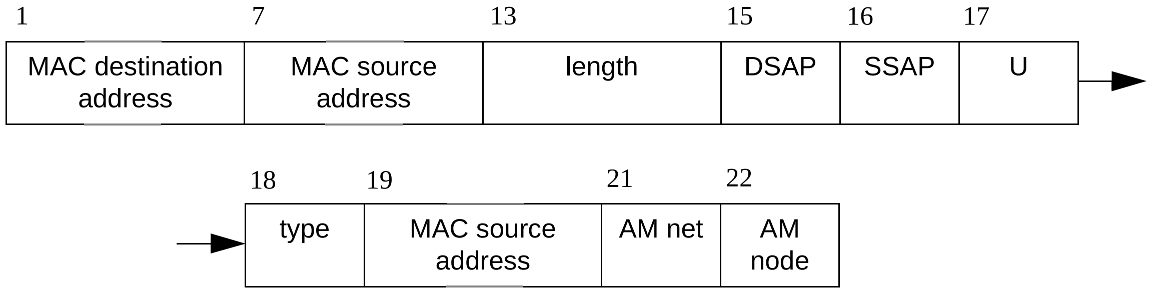
18	19	20	21
type prio	node	seq	MAC trailer

MAC destination address: **< 00, 00, 23, 00, 01, 01 >** (hexadecimal value)
MAC source address: **< 00, 00, 23, 00, 19, 01 >** (hexadecimal value)
Length: **6**
DSAP: **14** (hexadecimal value)
SSAP: **14** (hexadecimal value)
U: **3** (always)
Type/prio: bit 0-3: **1** (acknowledge message)
bit 4-7: **3**(normal priority)
node: **19**

seq: the received sequence counter increased by one
MAC trailer: padding and checksum

3.4.1 "I am here" message

To demonstrate to other stations on the GCOM subnet that the external computer is active, the "I am here" messages should be sent every second. At the same time, the MasterGate is sending its "I am here" messages. For further description of the signal, see Section 1.8.4 Transit Handling.



The "I am here" message from the external computer appears as follows:

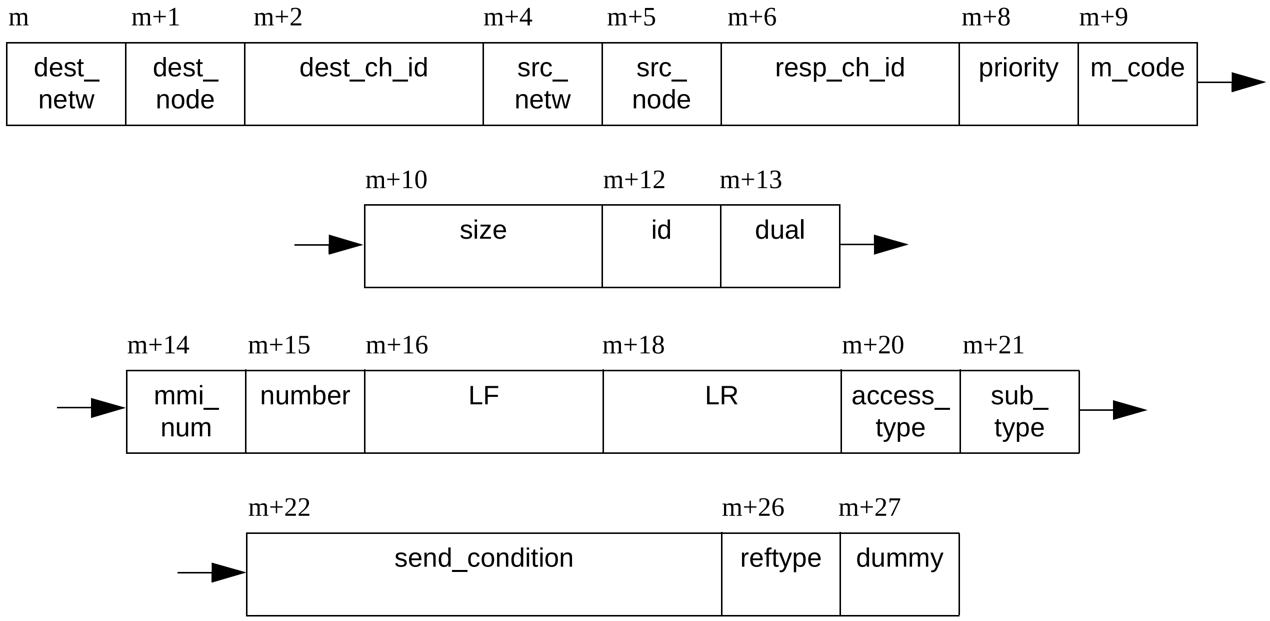
MAC destination address: **< 01, 00, 23, 00, 00, 01 >** (hexadecimal value)
MAC source address: **< 00, 00, 23, 00, 19, 01 >** (hexadecimal value)
Length: **12**
DSAP: **10** (hexadecimal value)
SSAP: **10** (hexadecimal value)
U: **3** (always)
Type: **4** ("I am here" message) (Note that bit 4-7 (prio) is not used! Set to 0.)
MAC source address: **< 00, 00, 23, 00, 19, 01 >** (hexadecimal value)
AM net: **110**
AM node: **19**

The "I am here" message from the MasterGate appears as follows:

MAC destination address: **< 01, 00, 23, 00, 00, 01 >** (hexadecimal value)
MAC source address: **< 00, 00, 23, 00, 01, 01 >** (hexadecimal value)
Length: **12**
DSAP: **10** (hexadecimal value)
SSAP: **10** (hexadecimal value)
U: **3** (always)
Type: **4** ("I am here" message) (Note that bit 4-7 (prio) is not used! Set to 0.)
MAC source address: **< 00, 00, 23, 00, 01, 01 >** (hexadecimal value)
AM net: **110**
AM node: **1**

3.4.2 Communication Status

The DSX signal for a request for communication status appears as follows. (For further description of the signal, see Section 3.5.5.3 Communication Status.)



Address part:

dest_netw: **110**
dest_node: **1**
dest_ch_id: **29**
src_netw: **110**
src_node: **19**
resp_ch_id: **26**
priority: **3** (always for external computer)
m_code: **0** (not used)
size: **16**
id: **2**
dual: **0** (not used)

Data part:

mmi_num: **1**
number: **1**
LF: **29344**
LR: **0**
access_type: **0**
sub_type: **0**
send_condition: **0**
ref_type: **18**

The answer from the MasterGate:

m	m+1	m+2	m+4	m+5	m+6	m+8	m+9
dest_netw	dest_node	dest_ch_id	src_netw	src_node	resp_ch_id	priority	m_code

m+10	m+12	m+13
size	id	dual

m+14	m+15	m+16	m+17	m+18	m+20
mmi_num	number	ref_type	dummy	LF	LR

m+22	m+26	m+27	m+28	m+30
status	netw	node	LF	LR

m+32	m+36	m+37	m+38	m+40
status	netw	node	LF	LR

m+42	m+46	m+47
status	netw	node

Address part:

dest_netw: **110**
dest_node: **19**
dest_ch_id: **26**
src_netw: **110**
src_node: **1**
resp_ch_id: **0** (not used)
priority: **3**
m_code: **0** (not used)
size: **306** (the size of the data part is always 304 bytes, despite the number of nodes)
id: **105**
dual: not used

Data part:

mmi_num: **1**
number: **3** (netw/node: 110/1; 11/15 and 11/16)
ref_type: **18**

First connection (node 1):

LF: no useful information
LR: no useful information
status: **2048** (GCOM link)
netw: **110**
node: **1**

Second connection (node 15):

LF: no useful information
LR: no useful information
status: **128** (MB300 link)
netw: **11**
node: **15**

Third connection (node 16):

LF: no useful information
LR: no useful information
status: **128** (MB300 link)
netw: **11**
node: **16**

3.4.3 Dataset Request

The DSX signal for a request for dataset no. 1 from node 15 appears as follows. (For further description of the signal, see Section 3.5.4.1 Data Set Communication (DSC).)

m	m+1	m+2	m+4	m+5	m+6	m+8	m+9
dest_ netw	dest_ node	dest_ch_id	src_ netw	src_ node	resp_ch_id	priority	m_code

m+10	m+12	m+13	m+14	m+15
size	dsd	netw	node	stat

Address part:

dest_netw: **11**
dest_node: **15**
dest_ch_id: **2**
src_netw: **110**
src_node: **19**

resp_ch_id: **1**
priority: **3** (always for external computer)
m_code: **1** (dataset no 1)
size: **4**

Data part:

dsd: **1**
netw: **110**
node: **19**
stat: **1** (always)

If dataset no. 1 contains four DAT elements with values 1, 2, 3 and 4, the DSX signal in the answer from the controller looks like this. (For further description of the signal, see Section 3.5.4.1 Data Set Communication (DSC).)

m	m+1	m+2	m+4	m+5	m+6	m+8	m+9
dest_ netw	dest_ node	dest_ch_id	src_ netw	src_ node	resp_ch_id	priority	m_code

m+10	m+12	m+16
size	value 1	value 2

m+20	m+24
value 3	value 4

Address part:

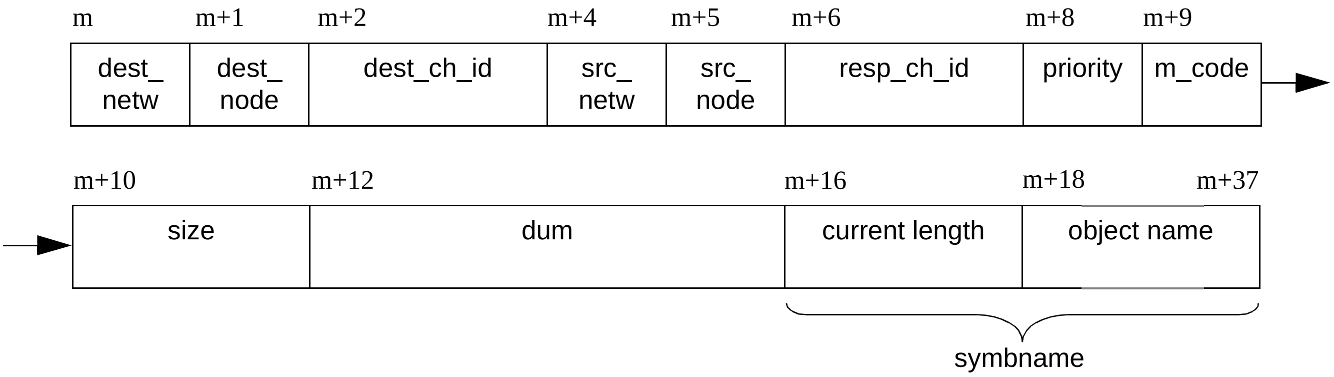
dest_netw: **110**
dest_node: **19**
dest_ch_id: **1**
src_netw: **11**
src_node: **15**
resp_ch_id: **0** (not used)
priority: **4**
m_code: **1**
size: **16** (the size of the data part is 4 bytes*no of values)

Data part:

value1: **1**
value2: **2**
value3: **3**
value4: **4**

3.4.4 Symbolic Name Translation

A symbolic name translation is performed to obtain the logical database reference for the object AIC16_1, which you need when you request a subscription. The DSX signal in the request appears as follows. (For further description of the signal, see Section 3.5.5.1 Symbolic Name Translation.)



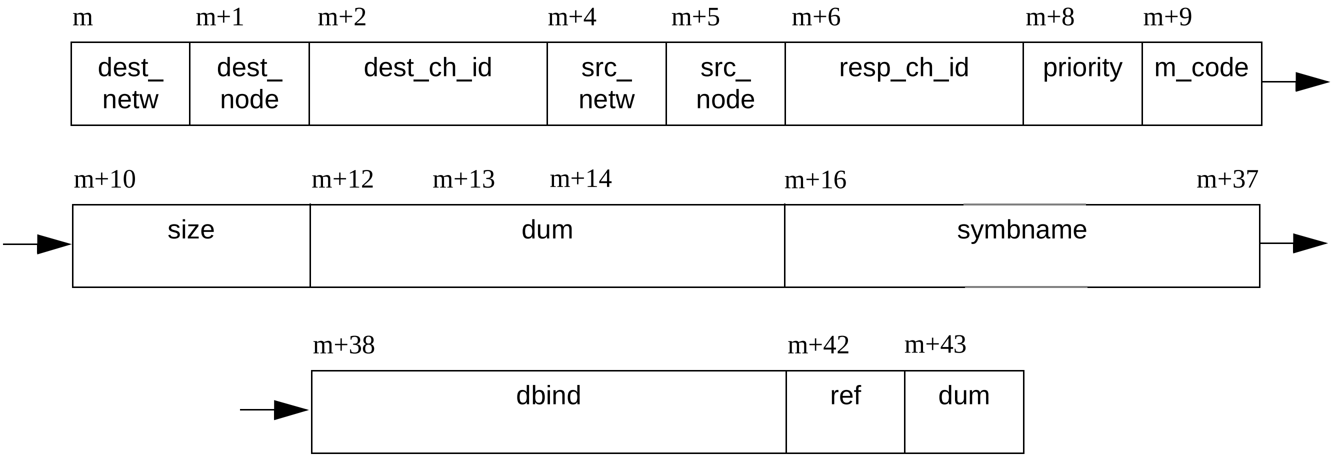
Address part:

dest_netw: **11**
dest_node: **16**
dest_ch_id: **49**
src_netw: **110**
src_node: **19**
resp_ch_id: **80**
priority: **3** (always for external computer)
m_code: **0** (not used)
size: **26**
(id and dual not used)

Data part:

dum: free to use, returned in the answer.
symbname:
current length: **7**
object name: **< 41, 49, 43, 31, 36, 5F, 31 >** (hexadecimal value)

The following answer is received:



Address part:

dest_netw: **110**
dest_node: **19**
dest_ch_id: **80**
src_netw: **11**
src_node: **16**
resp_ch_id: **0** (not used)
priority: **3**
m_code: **0** (not used)
size: **32**

Data part:

dum: returned from the request.
symbname:
    current length: **7**
    object name: **< 41, 49, 43, 31, 36, 5F, 31 >** (hexadecimal value)
dbind:
    LF: **6**
    LR: **1**
ref: **3**

3.4.5 A Data Subscription Request (demand)

The data subscription is accomplished with the signal sub_req. (For further description of the signal, see Section 3.5.3.1 Data Subscription.)

m	m+1	m+2	m+4	m+5	m+6	m+8	m+9
dest_ netw	dest_ node	dest_ch_id	src_ netw	src_ node	resp_ch_id	priority	m_code

m+10	m+12	m+13	m+14		m+16	m+37
size	id	dual	mmi	number	LF	LR

m+38	m+39	m+40	m+44	m+45
access_ type	sub_ type	send_condition	ref_type	dummy

Address part:

dest_netw: **11**
dest_node: **16**
dest_ch_id: **29**
src_netw: **110**
src_node: **19**
resp_ch_id: **26** (mmi=1)
priority: **3** (always for external computer)
m_code: **0** (not used)
size: **16**
id: **2** (sub_req)
dual: **0** (not used)

Data part:

mmi: **1**
number: **1** (the number of objects in the request)
LF: **6** (received in the symbolic name translation answer)
LR: **1** (received in the symbolic name translation answer)
access_type: **1** (ain_ab1_dcx for AI a_cupdate)
sub_type: **1** (demand data subscription)
send_condition: **0** (not used)
ref_type: **3** (ai)

The answer on the request follows. (For further description of the signal, see Section 3.5.3.2 Answer on Data Subscription.)

m	m+1	m+2	m+4	m+5	m+6	m+8	m+9
dest_ netw	dest_ node	dest_ch_id	src_ netw	src_ node	resp_ch_id	priority	m_code

m+10	m+12	m+13	m+14		m+16		m+28
size	id	dual	mmi	number	type	dummy	object dependent data

Address part:

dest_netw: **110**
dest_node: **19**
dest_ch_id: **26** (mmi=1)
src_netw: **11**
src_node: **16**
resp_ch_id: **0** (not used)
priority: **2**
m_code: **0** (not used)
size: **20**
id: **14** (a_cupdate)
dual: **0** (not used)

Data part:

mmi: **1**
number: **1** (the number of objects in the answer)
type: **3** (ai)
object dependent data:
  LF: **6**
  LR: **1**
  status: **5D** (hexadecimal value)
  value: **10.00** (in hexadecimal **<41 20 00 00>**)
  nodec: **2**

3.5 Application Procedures

When you add an external computer on a Master network, the new node must show that it exists. To do so, it sends “I am here” messages every second. When other nodes receive these messages, they note that the external computer is active. The second thing it does is check the status of the other nodes with a communication request, so information is collected about which nodes exist in the network. Then data collection can be done using data set, subscription, order or TTD.

3.5.1 Application Layer

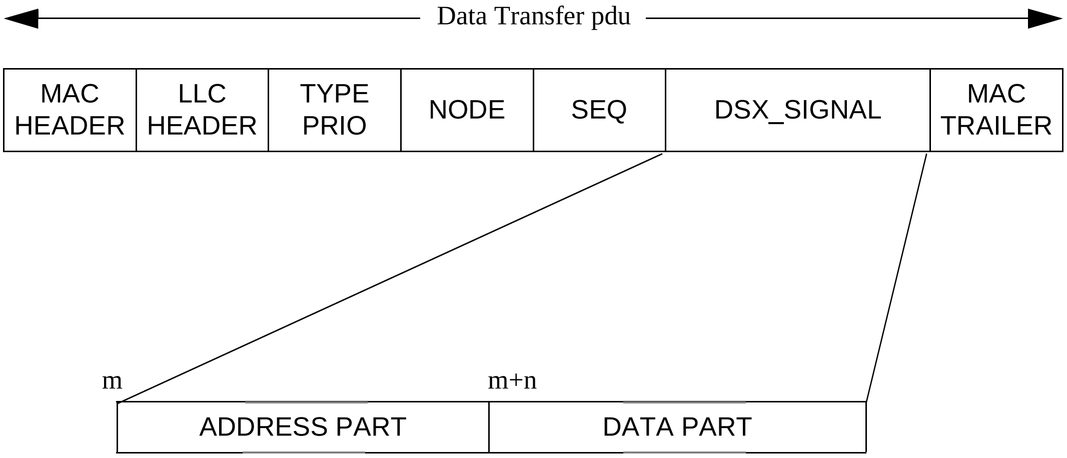


Figure 3-2. DSX signal

The GCOM communication subnet allows user tasks to send messages between stations, connected to the same IEEE 802.3 bus within an ABB Master Control Network or Plant Network.

With the transit function in MasterNet, a user task on the GCOM subnet can send and receive signals to/from any ABB Master station in a MasterNet Control Network or Plant Network.

A DSX signal to or from the ABB Master system is divided into two parts:

- Address part
- Data part

n = length of address part. n=12 for symbolic name translation to external computers, DSC and time synchronization messages. n=14 for all other messages.

3.5.2 Address Part

The structure of the address part (dsx-header) looks like this:

m	m+1	m+2	m+4	m+5	m+6	m+8	m+9	m+10	m+12	m+13

m m+1 m+2	dest_netw dest_node dest_ch_id	intb intb intw
m+4 m+5 m+6	src_netw src_node resp_ch_id	intb intb intw
m+8 m+9 m+10	priority m_code size	czprio intb intw
m+12 m+13	id dual	intb intb

NOTE

In symbolic name translation to an external computer, DSC and time synchronization messages **id** and **dual** are not a part of the address part (number of bytes in the address part is 12).

The address part of a message can be divided into four parts:

- Destination address
- Source address
- Message information
- Signal identity (except for symbolic name translation, DSC and time synch messages).

3.5.2.1 Messages to ABB Master System.

Destination address

Identifies the addressed function and consists of the following:

dest_netw	Destination network (11-127). Defines the network number in which the receiving function is located.
dest_node	Destination node (1-99,127). Defines the node number in which the receiving function is located.
dest_ch_id	Destination channel index. Specifies the function type in receiving node. See description of each function and Appendix B.

Source address

Identifies the source for the message and consists of the following:

src_netw	Source network (11-119). Defines the external computer’s logical network number in the ABB Master system.
src_node	Source node (10-99). Defines the external computer’s logical node number in the ABB Master system.
resp_ch_id	Response channel index. Defines the dest_ch_id for the response message and is predefined for each function. See description of each function and Appendix B.

The source address is used as the destination address for a response from the ABB Master system to the external computer.

Message information

Defines the message priority, message code and size.

priority	Message priority. Always 3 for external computers.
m_code	Message code. Used only in messages for DSC, TTD, MG-event subscription and in answer of symbolic name translation and order.
size	Size of data part including id and dual. The size must be even. If it is not, a dummy byte must be added to the end of the message.

Signal identity

Except for symbolic name translation to an external computer, DSC and time synchronization, an additional field (signal identity) specifies the content of the data part. For these signals, an additional field, dual, is also available (always set to 0).

id	Signal identity. See Appendix C.
dual	Dual computer (=0). Not used by the external computer.

For symbolic name translation to an external computer, DSC and time synchronization messages, only the dest_ch_id is used.

3.5.2.2 Messages to the External Computer

Destination address

Identifies the addressed function and consists of the following:

dest_netw	Destination network (11-99; 110 -119). Defines the external computer’s logical network number.
dest_node	Destination node (10-99). Defines the external computer’s logical node number.
dest_ch_id	Destination channel index. Specifies the function type. See description of each function and Appendix B.

Source address

Identifies the source of the message and consists of the following:

src_netw	Source network (11-127). Defines the ABB Master unit’s logical network number.
src_node	Source node (1-99,127). Defines the ABB Master unit’s logical node number.
resp_ch_id	Response channel index. Specifies the dest_ch_id in the response message.

The source address is used as the destination address for a response from the external computer to the ABB Master system.

Message information

Defines the message priority, message code and size.

priority	Message priority.
m_code	Message code. Used only in messages for DSC, TTD and MG-event subscription.
size	Size of data part including id and dual. The size must be even. If it is not, a dummy byte must be added to the end of the message.

Signal identity

Except for symbolic name translation to an external computer, DSC and time synchronization, an additional field (signal identity) specifies the content of the data part. For these signals, an additional field, dual, is also available (always set to 0).

id	Signal identity. See Appendix C.
dual	Dual computer (=0). Not used by the external computer.

For symbolic name translation to an external computer, DSC and time synchronization messages, only the dest_ch_id is used.

3.5.3 Data Part Object Functions

With the object function, the external computer can work directly against objects in the ABB Master system.

In the ABB Master system, there is one object handler and one data subscription handler for each object type (reference type). The reference type is used to identify an object type in a subscription. They are:

Table 3-1. ABB Master reference types

reference type	object
ain_dcx = 3	analog input
aout_dcx = 4	analog output
din_dcx = 5	digital input
dout_dcx = 6	digital output
controller_dcx = 13	PIDCON
Manual_stn_dcx = 25	MANSTN
ratio_stn_dcx = 26	RATIOSTN

Logical File (LF) and Logical Record (LR) are the logical database references (refadr) for an object in the ABB Master system. See also Section 3.1.3 System Functions.

When an external computer works against objects in the ABB Master system, it defines itself with user numbers (mmi number). Allowed mmi numbers for external computers on a GCOM subnet are mmi numbers 1 and 2.

Through the mmi number, the response channel identity (resp_ch_id) is defined:

Table 3-2. Response Channel Identity

mmi number	resp_ch_id
1	dcpc11
2	dcpc12

An object function message includes an address part and a data part. Each object function message is identified by the dest_ch_id and signal identity (id).

An object function message is addressed to the node (unit) where the object is located.

The external computer receives a system text message (syst_text) if it makes an illegal order, data subscription or TTD function.

The object functions are:

- **Data subscription for objects.** Data subscription is a function that collects one or several specified properties (parameters) from one or several specified database references (objects). A subscription request message can consist of several object types from the same ABB MasterPiece or Advant Controller. The collected properties are sent in different response messages, depending on object type (reference type).
- **Order.** With the function order, the external computer can control and change parameters for objects in the ABB Master system. Objects which are continuously controlled by external computers should be handled by the DSC function.
- **Data subscription for Time Tagged Data (TTD)**. External computers can subscribe to historical Time Tagged Data (TTD) from the ABB MasterPieces or Advant Controllers.

3.5.3.1 Data Subscription

Data subscription is a function that collects one or several specified properties (parameters) from one or several specified database references (objects). A subscription request message can consist of several object types from the same ABB MasterPiece or Advant Controller. The collected properties are sent in different response messages, depending on object type (reference type).

In an ABB MasterPiece or Advant Controller, there is one data subscription handler for each object type. A subscription handler for an object type can handle data subscriptions from up to 16 users (mmi numbers) at the same time, where one user, for example, may be one screen on a MasterView. Each of the users can work at the same time with several objects. If a subscription handler is busy (more than 16 users to handle), it sends a message (syst_text) to the user.

NOTE

> It is important to limit the number of external subscribers to avoid locking out object access by the operator.

With this function, external computers can work against the following object types in the ABB Master system:

- Analog input
- Analog output
- Digital input
- Digital output
- PIDCON (Process controller)
- RATIO station (Ratio controller)
- MANUAL station (Reference generator)

A data subscription is activated by a SUB_REQ message:

dest_ch_id=DCSC00
id=sub_req

The message contains user identity (mmi_number) and references to one or several objects, each one with an individual access type and subscription type.

The access type describes which properties are to be collected for the object and the subscription type describes what kind of subscription it is: demand, on event or cyclic.

The properties to be collected from the ABB MasterPiece or Advant Controller database for an object are described with an access type. The access type for on event and cyclic data subscriptions is optimized for each object type in the ABB Master system and cannot be controlled by the external computer. The cyclic signals contain everything that can be changed by the process and everything not detected as events by other functions (object handlers).

Allowed access types are:

Table 3-3. Allowed Access types

object	access type	signal identity in answer
Analog input	ain_ab1_dcx=1 ain_ab2_dcx=2 ain_ab3_dcx=3 ain_ab4_dcx=4	a_cupdate na_demand la_demand aiDevDupdate
Analog output	aout_ab1_dcx=1 aout_ab2_dcx=2 aout_ab3_dcx=3	a_cupdate na_demand aoDevDupdate
Digital input	din_ab1_dcx=1 din_ab2_dcx=2	d_eupdate dDevDupdate
Digital output	dout_ab1_dcx=1 dout_ab2_dcx=2	d_eupdate dDevDupdate
PIDCON	ctrl_ab1_dcx=1 ctrl_ab2_dcx=2 ctrl_ab3_dcx=3	pid_cupdate pid_n_dupdate pid_l_dupdate
RATIO station	rs_ab1_dcx=1 rs_ab2_dcx=2 rs_ab3_dcx=3	rs_cupdate rs_n_dupdate rs_l_dupdate
MANUAL station	ms_ab1_dcx=1 ms_ab2_dcx=2 ms_ab3_dcx=3	ms_cupdate ms_n_dupdate ms_l_dupdate

There are three types of data subscription: demand (single shot), on event and cyclic.

Allowed subscription types (sub_type) are:

Table 3-4. Allowed Subscription Types

subscription type	**description**
dummy_up_dcx = 0	dummy
demand_up_dcx = 1	demand data subscription
event_up_dcx = 2	event data subscription
cycle1_up_dcx = 3	cyclic data subscription (1 sec)
cycle2_up_dcx = 5	cyclic data subscription (3 sec)
cycle3_up_dcx = 6	cyclic data subscription (9 sec)

A demand data subscription gives a single shot update from one or several objects. The access type for the demand update is user controlled in the request message.

An event data subscription first gives a demand (current state) and, thereafter, an on event update from one or several objects. The access type for the demand update is user controlled in the request message. The access type for an on event update depends on the object type:

Table 3-5. Access type for on event update

object type	**access type**
AI	ain_ab3_dcx
AO	aout_ab3_dcx
DI	din_ab2_dcx
DO	dout_ab2_dcx
PIDCON	ctrl_ab2_dcx
RATIOSTN	rs_ab2_dcx
MANSTN	ms_ab2_dcx

For a definition of the conditions for an event from an object, see Section 3.5.3.4 Order. An event is defined as a status change of the object. Also see Section 3.6.2.2 Status for AI and AO Objects, Section 3.6.2.4 Status for DI and DO Objects and Section 3.6.2.6 Parameters and Status for PIDCON, RATIO and MANSTN for descriptions of the object status information.

A cyclic data subscription first gives a demand update and, thereafter, on event and cyclic updates. The access type for the demand update is user controlled in the request message. The access type for cyclic update depends on the object type:

Table 3-6. Access type for cyclic update

object type	access type
AI	ain_ab1_dcx
AO	aout_ab1_dcx
DI	din_ab1_dcx
DO	dout_ab1_dcx
PIDCON	ctrl_ab1_dcx
RATIOSTN	rs_ab1_dcx
MANSTN	ms_ab1_dcx

Deactivation is accomplished with the message sub_clear

dest_ch_id = DCSC00
id = sub_clear

To deactivate a selected set of objects, the message must contain the same information as the sub_req message activating the subscription.

Deactivation can also be accomplished with the message mmi_clear

dest_ch_id = DCSC00
id = mmi_clear

The message clears all data subscriptions for the mmi_number.

The communication link between MasterGate and the MasterPiece or Advant Controllers is supervised and, if the connection is lost with a MasterPiece or Advant Controller, every data subscription from external computers connected to the MasterGate is cleared.

The external computer clears all current (outstanding) on event and cyclic type data subscriptions at restart.

sub_req and sub_clear messages

A sub_req or sub_clear message can contain up to 16 objects. Number is the number of objects in the message. For each object, the part starting with LF and ending with dumsize is repeated. Dumsize is not included for the last object. The message is addressed to the unit (end node) where data are located and is identified by dest_ch_id and id.

m	n	n+1	n+2	n+4	n+6	n+7
address part	mmi	number	LF	LR	access_ type	sub_ type

n+8	n+12	n+13	n+14	
send_condition	ref_ type	dummy	dumsize	LF

- **mmi:** mmi number (1 or 2)
- **number** is the number of objects in the message
- **LF:** logical file for the object
- **LR:** logical record for the object
- **access_type:** see
- **sub_type:** see Table 3-4
- **send_condition** must always be zero (not used by the external computer)
- **ref_type:** see Table 3-1.

Sub_req and sub_clear data part length 16..224 bytes.

For Pascal layout, see Section 3.6.1.1 Pascal Layout Subscription Request and Clear.

mmi_clear message

The message is addressed to the unit (end node) where data are located and is identified by dest_ch_id and id.

	n	n+1	n+2 n+3
address part	mmi_num	dummy	dumsize

- **mmi_num:** mmi number (1 or 2).

mmi_clear data part length 4 bytes.

For Pascal layout, see Section 3.6.1.2 Pascal Layout mmi clear.

3.5.3.2 Answer on Data Subscription

An answer on a sub_req can contain from one to 38 objects of the same object type (depending on object type and access type). For each object, the part starting with object-dependent data and ending with dumsize is repeated. Dumsize is not included for the last object.

The message is identified by the dest_ch_id and id

dest_ch_id = resp_ch_id in the request (see Table 3-2)
id = signal identity, depends on object type and access type in the sub_req (see Table 3-3 and Appendix C).

m	n	n+1	n+2	n+3	n+4	
address part	mmi	number	type	dummy	object dependent data	dumsize

- **mmi:** mmi number (1 or 2)
- **number** is the number of objects in the message
- **type** is the reference type, see Table 3-1
- **object dependent data:** see Pascal layout, Section 3.6.2 Answer on Data Subscription.

Table 3-7. Maximum number of references/signal identity

max no. of objects	signal identity
max_deup = 38	d_eupdate
max_acup = 23	a_cupdate
max_nad = 4	na_demand
max_lad = 3	la_demand
max_pidcup = 4	pid_cupdate
max_pidndup = 2	pid_n_dupdate
max_pidldup = 1	pid_l_dupdate
maxDDev = 5	dDevDupdate
maxAoDev = 4	aoDevDupdate
maxAiDev = 3	aiDevDupdate
max_rscup = 4	rs_cupdate
max_rsndup = 2	rs_n_dupdate
max_rsldup = 1	rs_l_dupdate
max_mscup = 7	ms_cupdate
max_msndup = 3	ms_n_dupdate
max_msldup = 2	ms_l_dupdate

The parameters subsystem, class and value_treat in the object-dependent data part should not be used by the external computer.

Data part length 12..386 bytes. For Pascal layout and status, see Section 3.6.2 Answer on Data Subscription.

3.5.3.3 Historical Time Tagged Data (TTD)

External computers can subscribe to historical Time Tagged Data (TTD) from the ABB MasterPieces or Advant Controllers.

A TTD log is a list of regular read (sampled) values of a process signal or calculated signal. An historical time tagged data subscription can only be of demand type (single shot).

The external computer can also, with the signal ttd_request (opcode= getttdinf_dcx), get a translation from TTD log variable name to logical database reference in the ABB MasterPieces or Advant Controllers. (Opcode = operation code which defines the data part in the ttd_request signal. See data part for TTD.)

TTD log group: A TTD log group is a logical collection of values which are separated in time by a constant period, the log interval.

In one TTD log group, you can have a maximum of 127 log variables (process signals). Log intervals are normally different for each log group (standard four log groups), but they can also be the same for many TTD log groups.

TTD log variables: A logged variable can be of different data types, i.e., real, intl, intw or boolean. The values for the logged variable are stored in a database file. This file is circular, which means that when the file is full, the oldest value is overwritten by the youngest. The logging intervals are normally:

- 15 seconds
- Two minutes
- Six minutes
- One hour.

The following process signals and corresponding properties within a process signal can be stored in a TTD log:

- AI, AO, DI, DO (property = VALUE)
- PIDCON (property = MV, WSP, DEVIATION, POUT, PRES1)
- MANSTN (property = MV, POUT, PRES1)
- RATIOSTN (property = MV, WRATIO, OUT, PRES1)
- GENOBJ (property = MV, MV_INTL, OP)

A log variable can be primary logged or dependently logged. A primary logged variable is fetched from the database of the corresponding process signal. A dependent logged variable is fetched from another TTD log group with a shorter log interval. The log interval for a dependent log is always an even multiple of a TTD log with a shorter log interval.

TTD log functions: You can treat the log variable values in four different ways, as follows.

- **Momentaneous logging.** Momentaneous values from process signals are stored (primary logging).
- **Mean value logging.** A consecutive mean value is calculated and only the mean value is stored. Each value is sampled with a sampling interval that is shorter than the log interval and a mean value is calculated.
- **Max value logging.** Only the maximum value of a number of consecutive samples is stored.
- **Min value logging.** Only the minimum value of a number of consecutive samples is stored.

TTD Data Request

Subscription for TTD log variables is carried out by the signal ttd_request with opcode getttdval_dcx.

A ttd_request signal with opcode getttdval_dcx may contain a subscription for up to four variables and, for each variable, a maximum of 402 values can be requested.

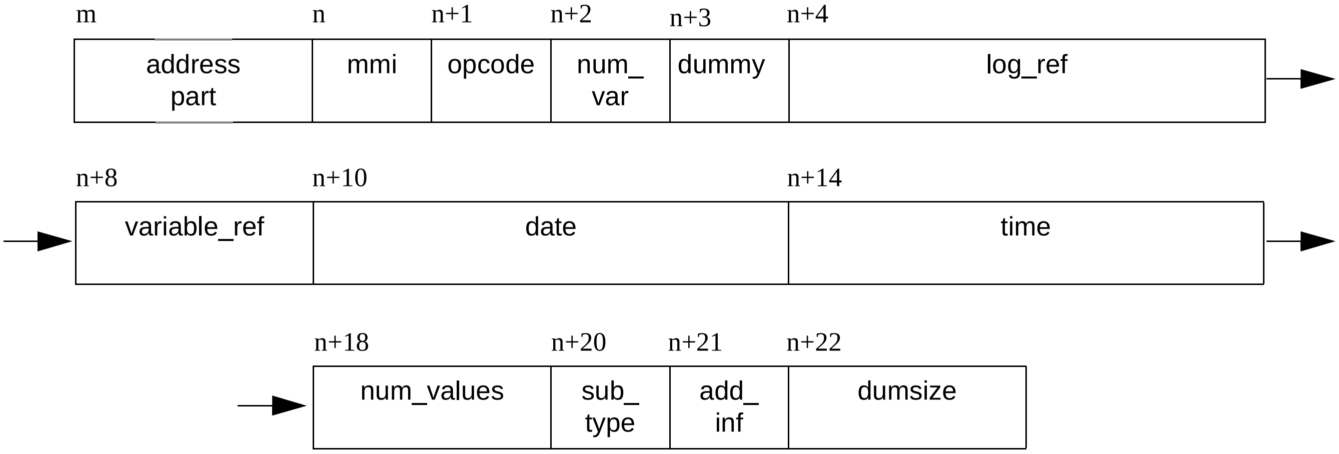
dest_ch_id = DCTC50
id = ttd_request
opcode = getttdval_dcx = 1

Num_var is the number of variables in the request. The external computer is not permitted to request more than one variable in each request.

For each variable, the part starting with log_ref and ending with dumsize is repeated. Dumsize is not included for the last object.

The message is addressed to the unit (end node) where data are located and is identified by dest_ch_id and id.

A ttd_request with opcode getttdval_dcx would look like this:



- **mmi:** mmi number (1 or 2)
- **opcode:** operation code, see Appendix C
- **num_var** is the number of requested variables, e.g., how many times the part started with log ref is repeated
- **log_ref** is database index for actual TTD log (log_interval)
- **variable_ref** is logged variable within actual TTD log (variable in the log interval)
- **date** is start date (number of days from 1980-01-01 binary coded with 1 january 1980 as day 1 (n+2 is the most significant byte)) for the first value (oldest value)
- **time** is start time (number of 0.1 ms from midnight) for the first value (oldest value)
- **num_values** is number of values from requested start date and start time. A maximum of 402 values can be requested in one request.
- **sub_type** is subscription type (only demand_up_dcx=1 is allowed)
- **add_inf** is a flag if additional information is requested. This is normally (always for external computers) set to TRUE (add_inf=1).

The data part length of ttd_request with opcode getttdval_dcx is 24..82 bytes.

TTD Data Answer

The answer from the TTD log function to the subscription request is sent in signal ttd_data. The signal contains logged values from one log variable. If more than 66 values are requested, the signal is divided into partsignals where each partsignal contains a maximum of 66 values. Each partsignal has a sequence number (field m_code in address part) which is updated for each partsignal.

For the last partsignal, a field in the signal (last_package) is set to indicate that there are no more values for this log variable.

dest_ch_id = resp_ch_id in the request (dcpc11 or dcpc12, see Table 3-2)
id = ttd_data.

TTD illegal values: Some logged values can be of type NAN (Not A Number). This can occur, for instance, when a process signal is error marked in the database due to hardware error on the corresponding I/O board. These values are still logged by the log function, but they are marked with a special bit combination that is not valid real, intl, intw or boolean values.

The following bit combination is implemented to denote NAN:

ilbadval_dcx = 2143289343; (* #7FBFFFFF real, intl values *)

iwbadval_dcx = 32703; (* #7FBF intw values *)

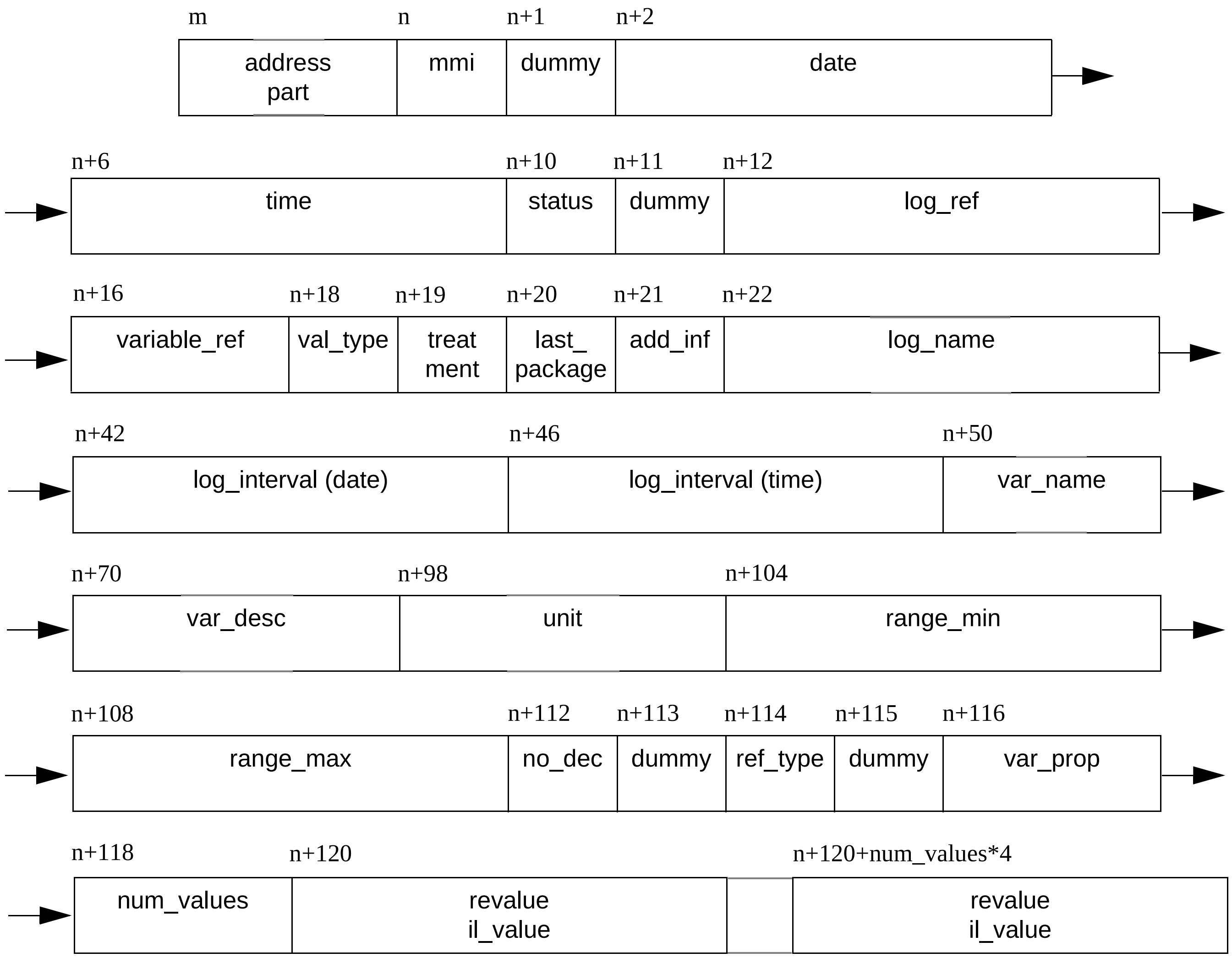
ibbadval_dcx = 32703; (* #7FBF boolean values *)

A ttd_data signal contains logged values from one logged variable together with log name, variable name, variable description and unit.

The message is identified by dest_ch_id and id.

If more than 66 values are requested in a ttd_request, then the values are divided in partsignals. Each partsignal has a sequence number (m_code in the address part), i.e., for the first partsignal, m_code=1, second, m_code=2, etc. A maximum of 402 values can be requested in a ttd_request.

For the last partsignal, last_package is set to 1.



- **mmi:** mmi number (1 or 2)
- **date** is start date (number of days from 1980-01-01 binary coded with 1 january 1980 as day 1 (n+2 is the most significant byte)) for the first value (oldest value).
- **time** is start time (number of 0.1 ms from midnight) for the first value (oldest value).

- **status** is a TTD request status byte with the value described below:

Table 3-8. TTD message status

value	description
dctdrqok = 0	ttd request OK
dctdilop = 1	illegal operation code in ttd request
dctdlgnf = 2	log not found
dctdlgni = 3	log not implemented
dctdvrnf = 4	variable not found
dctdvrni = 5	variable not implemented
dctdvror = 6	number of requested variables out of range
dctdvlor = 7	number of requested values out of range
dctdtorc = 8	time(s) out of range, completely
dctdtoro = 9	time(s) out of range, oldest limit
dctdtory = 10	time(s) out of range, youngest limit
dctdtorb = 11	time(s) out of range, both limits
dctdlgna = 12	logging not active
dctdlgai = 13	logging already inactive
dctdlgaa = 14	logging already active
dctdmlna = 15	mass memory logging not active
dctdmlai = 16	mass memory logging already inactive
dctdmlaa = 17	mass memory logging already active
dctdillf = 18	illegal reference of logical file number
dctdilst = 19	illegal subtype request
dctdilvt = 20	illegal value type
dctdilti = 21	illegal value of time
dctdlgnu = 22	log not user type
dctdilre = 23	illegal reference type
dctdilpr = 24	illegal property number
dctdnain = 25	no additional info defined
dctdvdlg = 26	variable is dependently logged
dctdrcnd = 27	referenced concept not defined in system
dctdrfnf = 28	referenced file not found
dctdrfni = 29	referenced file not implemented

Table 3-8. TTD message status

dctdrvnf = 30 dctdrvni = 31 dctdrvin = 32 dctdapvf = 33 dctdiprr = 34 dctdfgad = 35 dctdbupl = 36 dctdproe = 37 dctdilln = 38 dctdilsi = 39	referenced variable not found referenced variable not implemented referenced variable not initiated access to primary variable failed illegal property reference for primary variable failed to get physical address log is backup log program error in TTD illegal log number illegal sample interval
dctdrlus = 40 dctdrlbu = 41 dctdiimo = 42 dctdlrli = 43	referenced log is user log referenced log is backup log illegal geti_mode neither log_ref nor log_interval was given

- **log_ref** is database index (LF, LR) for actual TTD log.
- **variable_ref** is logged variable within actual TTD log.
- **val_type** is the storage type (data type) for the log variable values. The following val_types are implemented:

val_type	description
retype = 1	real value
iltype = 2	integer long value
iwtype = 3	integer word value
ibtype = 4	integer byte value

- **treatment** is the type of treatment for the log variable values. The following treatment is implemented:

treatment	description
mom_ttd = 0	momentaneous logging
mean_ttd = 1	mean value logging
max_ttd = 2	max value logging
min_ttd = 3	min value logging

- **last_package** is set to 1 when the last partsignal is sent from TTD; otherwise, it is set to 0.
- **add_inf** is set to 1 if additional information is requested in corresponding ttd_request.

- **log_name** is a symbolic text for the log name which is unique in a node. It is an array of 20 characters.
- **log_interval** (date) is the “day” portion of the time interval between two consecutively stored variable values in a log.
- **log_interval** (time) is the “time” portion (0.1 ms) of the time interval between two consecutively stored variable values in a log.
- **var_name** is the variable name as given in the original process database file (AI, AO, etc.) for the object. It is an array of 20 characters.
- **var_descr** is the variable description as given in the original process database file (AI, AO, etc.) for the object. It is an array of 28 characters.
- **unit** is the unit as given in the original process database file (AI, AO, etc.) for the object. It is an array of six characters.
- **range_min** is the minimum value of the log variable as given in the original process database for the object.
- **range_max** is the maximum value of the log variable as given in the original process database for the object.
- **no_dec** is the number of decimals for the log variable as given in the original process database for the object.
- **ref_type** is the reference type for the log variable as given in the original process database for the object.
- **var_prop** is the property number within actual concept for the log variable as given in the original process database for the object.
- **num_values** is the number of values from the date and time above. Num_values can be in the range 0-66.
- **revalue** is the start for the actual logged values. A maximum of 66 values can be put in one partsignal. The size for each value is four bytes.

The data part length of ttd_data is 120..388 bytes.

For Pascal layout, see Section 3.6.3.2 Pascal Layout for Data Answer.

TTD Information Request

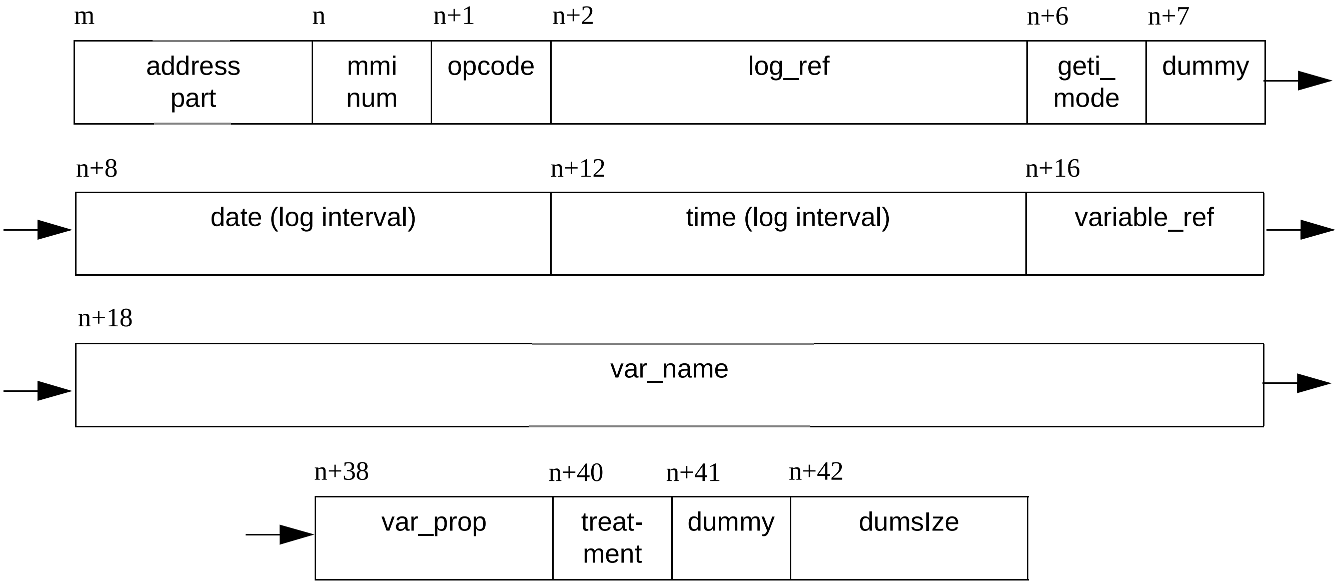
Information for TTD log variables is requested with the signal ttd_request with opcode getttdinf_dcx. A ttd_request signal with opcode getttdinf_dcx generates a search for the requested TTD log variable and a fetch of information.

dest_ch_id = DCTC50
id = ttd_request
opcode = getttdinf_dcx = 2

A ttd_request with opcode getttdinf_dcx contains a variant field which makes it possible to select different modes to use when the information is fetched. This means that you can specify the wanted variable by index or make a search for the variable either by the name of the variable or by the variable's database index.

The external computer addresses the message to the unit (end node) where the variable is located.

A ttd_request with opcode getttdinf_dcx, used when you want information for a logged variable by name of the variable, looks like this:



- **log_ref** is the database index for the actual TTD log. (80,0 search in all logs.)
- **geti_mode** should be equal to 1 (sea_name). (Use this mode when the information is fetched.)
- **date** is the date part of the log interval for the actual TTD log. (Usually set to 0 if not log interval > one day.)
- **time** is the time part of the log interval for the actual TTD log (in number of 0.1 ms from midnight). See part Historical time tagged data (TTD log group).
- **variable_ref** should be set to 0 (for search in all log variables).

- **var_name** is the name of the variable object.
- **var_prop** is the property number of the logged property in the original database file of the object.

Table 3-9. Property number

object	property	prop no
AI	VALUE	19
AO	VALUE	10
DI	VALUE	12
DO	VALUE	12
PIDCON	MV WSP DEVIATION POUT PRES1	78 80 81 86 159
MANSTN	MV POUT PRES1	48 52 98
RATIOSTN	MV WRATIO OUT PRES1	57 59 60 120

- **treatment** is the log function used for the variable. See ttd_infdata.

The data part length of ttd_request with opcode getttdinf_dcx is 44 bytes.

For Pascal layout, see Section 3.6.3.1 Pascal Layout for Subscription.

TTD Information Answer

The answer from the TTD log function to the request for information is sent in signal ttd_infdata. The signal contains all information for one log variable.

dest_ch_id = resp_ch_id in the request (dcpc11 or dcpc12, see Table 3-2)
id = ttd_infdata.

A ttd_infdata signal contains information concerning one logged variable in a TTD log. The message is identified by dest_ch_id and id.

m	n	n+1	n+2	n+3	n+4
address part	mmi num	dummy	status	dummy	log_ref

n+8	n+28	n+32
log_name	date	time

n+36	n+38	n+58	n+86
variable_ref	var_name	var_desc	unit

n+92	n+96	n+100	n+101
range_min	range_max	no_of_dec	dummy

n+102	n+103	n+104	n+105	n+106	n+110	n+111
netw	node	ref_type	dummy	instance	primary	dummy

IF primary = TRUE THEN

n+112	n+114	n+116	n+118	n+120
var_property	test1_property	test2_property	rmin_property	rmax_property

n+122	n+124	n+126	n+127
nodec_property	unit_property	test1_logic	test2_logic

ELSE

n+112	n+114	n+116 n+127
var_property	var_property	12 dummys

n+128	n+129	n+130	n+131	n+132 n+135
val_type	treatment	bad_val_lim	scaling	scaling_c

- **status** is a TTD request status byte with the value described in Table 3-8.
- **log_ref** is the database index (LF,LR) for the actual TTD log. (Used for subscriptions.)
- **log_name** is a unique symbolic name for the log.
- **date** (log_interval) is the “day” portion (in days) of the time interval between two consecutively stored variable values in a log.
- **time** (log_interval) is the “time” portion (in 0.1 ms) of the time interval between two consecutively stored variable values in a log.
- **variable_ref** is the member (1-127) of the logged variable within the actual TTD log. (Used for subscriptions.)
- **var_name** is the symbolic name as given in the original process database file (AI, AO, etc.) for the object.
- **var_descr** is the description as given in the original process database file (AI, AO, etc.) for the object.
- **unit** is the unit as given in the original process database file (AI, AO, etc.) for the object.
- **range_min** is the minimum value of the log variable as given in the original process database for the object.
- **range_max** is the maximum value of the log variable as given in the original process database for the object.
- **no_dec** is number of decimals for the log variable as given in the original process database for the object.
- **network** is the network number of the source for the log variable.
- **node** is the node number of the source for the log variable.
- **ref_type** is reference type for the log variable, given by the original object type.
- **instance** is the database index (LF, LR) of the source for the log variable.
- **primary**: If TRUE (set to 1), the log variable is primary logged. If FALSE (set to 0), the log variable is dependently logged.

For a primary logged variable, use these fields:

- **var_property** is the property number of the value that is logged for the log variable as given within the actual concept of the original process database for the object.
- **test1_property** is the property number of the first boolean property that is to be tested at each sample. Not used by external computers.
- **test2_property** is the property number of the second boolean property that is to be tested at each sample. Not used by external computers.
- **rmin_property** is the property number of the range_min property.
- **rmax_property** is the property number of the range_max property.
- **nodec_property** is the property number of the no_dec property.
- **unit_property** is the property number of the unit property.

- **test1_logic** is the value that the test1_property is to be tested against at each sample. Not used by external computers.
- **test2_logic** is the value that the test2_property is to be tested against at each sample. Not used by external computers.

For a dependently logged variable, use these fields:

- **var_index** is the member (1-127) number of the variable that describes the source of the logged variable.
- **var_property** is the property number within actual concept for the log variable as given in the original process database for the object.
- **val_type** is the storage type (data type) for the log variable values. The following val_types are implemented:

val_type	description
retype = 1	real value
iltype = 2	integer long value
iwtype = 3	integer word value
ibtype = 4	integer byte value

- **treatment** is the type of treatment for the log variable values.The following treatments are implemented:

treatment	description
mom_ttd = 0	momentaneous logging
mean_ttd = 1	mean value logging
max_ttd = 2	max value logging
min_ttd = 3	min value logging

- **bad_val_lim** is the maximum number of sampled values that can be bad values without setting the logged value to a bad value. The limit is expressed in the percentage of values sampled for each value logged.
- **scaling** is true (set to 1) if the variable has a scaling factor defined.
- **scaling_c** is the scaling factor (of type real) with which the value from the object was multiplied when the logged value was created in TTD. This value is only relevant if scaling is true.

The data part length of ttd_infdata is 136 bytes.

For Pascal layout, see Section 3.6.3.3 Pascal Layout Log Variable.

3.5.3.4 Order

With the function order, the external computer can control and change parameters for objects in the ABB Master system. Objects which are continuously controlled by external computers should be handled by the DSC function.

External computers can control the following objects in the ABB Master system with the function order:

- Analog input (AI)
- Analog output (AO)
- Digital input (DI)
- Digital output (DO).

The following parameters can be changed for an object:

Table 3-10. Object parameters

object	parameter
analog input	value HI_LIM2 HI_LIM1 LO_LIM1 LO_LIM2 Block/deblock update of object Block/deblock alarm from object Acknowledge alarm
analog output	value Block/deblock update of object Block/deblock alarm from object Acknowledge alarm
digital input	value Block/deblock update of object Block/deblock alarm from object Acknowledge alarm
digital output	value Block/deblock update of object Block/deblock alarm from object Acknowledge alarm

If the external computer must control other objects, use the function DSC.

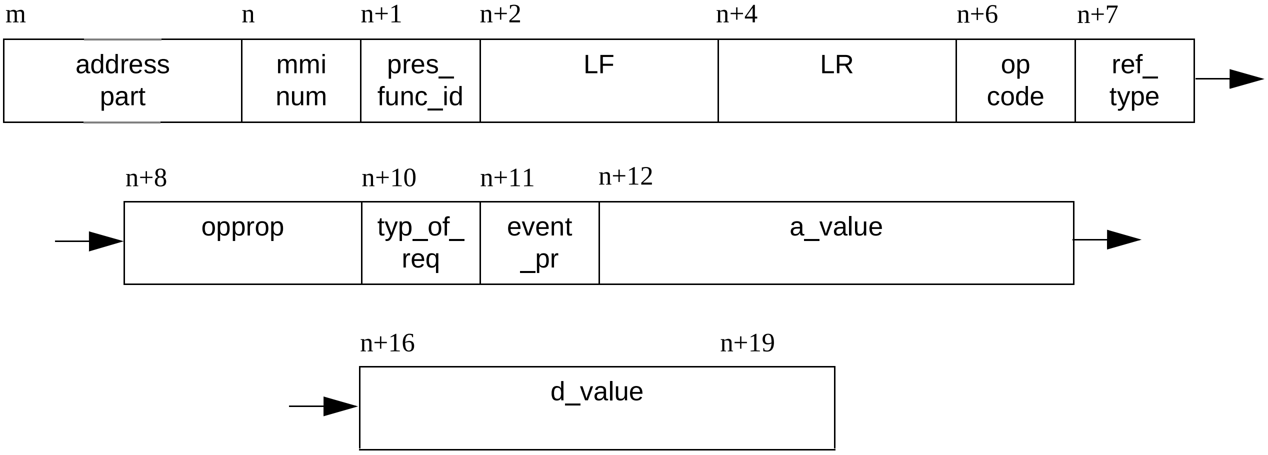
An order is given with the message order:

dest_ch_id = depends on the object type, see Appendix B
id = order

The type of order is defined with the operation code (opcode) and operation property (opprop). See Table 3-11 and Appendix D.

An order gives no response message (if the order is not illegal), so the function data subscription is used to check the result of an order.

The message is addressed to the unit (end node) where the object is located and is identified by dest_ch_id and id.



- **mmi:** mmi number (1 or 2)
- **pres_func_id:** 59+mmi_number
- **LF:** logical file for the object
- **LR:** logical record for the object
- **opcode:** see Table 3-11 and Appendix D.
- **ref_type:** see Table 3-1.
- **opprop:** see Table 3-11 and Appendix D.
- **typ_of_req** (intb): always equal to one (1) for external computer (1=program)
- **event_pr** (intb): always equal to one (1) for external computer.
- **a_value** (real): value or limit value for analog objects.
- **d_value** (intl): value (0 or 1) for digital objects.

Order data part length is 20 bytes.

For Pascal layout, see Section 3.6.4.1 Pascal Layout Order.

Manual input of value:

To change a value for an object, use the operation codes (opcode) and operation properties (opprop) as shown in Table 3-11. When an order to change a value is received, it is checked (including general controls) that the value is within the measure area (valid for analog) or it is 0/1 (valid for digital). If so, the new value is saved in the database. To give the value a correct status, a limit value control is initiated (valid for analog) or the new value is compared with the old (valid for digital). If an order is not accepted, a fault message (syst_text) is sent and the new value is not saved in the database.

For AI and DI objects, the process update must be blocked before a new value can be saved in the database.

Manual input of a limit value:

You can only change the alarm limits for analog inputs. The operation code (opcode) and operation properties (opprop) are shown in Table 3-11.

Before the order is executed, it is determined (beside general controls) that the new limit is inside the measure area. If not, a fault message (syst_text) is sent.

Event analyses for an object:

The external computer gets the following events from the objects if it has a data subscription of the type, on event or cyclic.

For an analog input object, the following are regarded as events:

- Passage of any low or high limit in any direction
- Coming and leaving signal faults
- Order, i.e., blocking of alarm, change of limit, etc.

Every limit passage is individually reported (i.e., if any reporting at all is done) even if several limits are passed at the same time.

Limit passages are detected by limit value controls that are performed when a new value is saved or a limit is changed.

For a digital input object, the following are regarded as events:

- Status change at the input (new value)
- Coming or leaving signal fault
- Order

Every status change is reported if this is not blocked.

For an analog output object, the following are regarded as events:

- Coming and leaving signal faults
- Order

Coming and leaving signal faults are detected when an output via the object handler is attempted.

For a digital output object, the following are regarded as events:

- Coming and leaving signal faults
- Order

Coming and leaving signal faults are detected when an output via the object handler is attempted.

Blocking/deblocking of object and event handling (alarms):

You can block/deblock the update of a value for an object and/or block/deblock the alarm handling for an object. The block/deblock is done individually for each object. To block the alarms from one object, set both printer (dist_print_dcx) and alarm (dist_alarm_block) (two orders).

Alarm acknowledgment:

The external computer can acknowledge all alarms for an object. See Table 3-11.

Syst_text message: See Section 3.5.3.5 System Text.

Order from an ABB Master Station to an external computer:

An order from an ABB Master Station is preceded by a select order (an order message with opcode = select_dcx and opprop = dummy_prop_dcx) and terminated with a deselect order (an order message with opcode = deselect_dcx and opprop = dummy_prop_dcx). The select order from an ABB Master Station must be answered with a select answer (sel_answer) message from the external computer. When the ABB Master Station has selected an object, no other user can control the object.

Table 3-11. Objects: opcode and opprop

object	parameter	opcode	opprop
analog input	Value	value_change_dcx	act_value_dcx
	HI_LIM2	value_change_dcx	hilim2_dcx
	HI_LIM1	value_change_dcx	hilim1_dcx
	LO_LIM1	value_change_dcx	lowlim1_dcx
	LO_LIM2	value_change_dcx	lowlim2_dcx
	Block/deblock alarms	block_dcx deblock_dcx block_dcx deblock_dcx	dist_print_dcx dist_print_dcx dist_alarm_dcx dist_alarm_dcx
	Block/deblock update of object	block_dcx deblock_dcx	proc_update_dcx proc_update_dcx
	Acknowledge alarm	acknowledge_dcx	edum_ev_prop
analog output	Value	value_change_dcx	output_dcx
	Block/deblock alarms	block_dcx deblock_dcx block_dcx deblock_dcx	dist_print_dcx dist_print_dcx dist_alarm_dcx dist_alarm_dcx
	Block/deblock update of object	block_dcx deblock_dcx	output_dcx output_dcx
	Acknowledge alarm	acknowledge_dcx	edum_ev_prop

Table 3-11. Objects: opcode and opprop

object	parameter	opcode	opprop
digital input	Value	value_change_dcx	act_value_dcx
	Block/deblock alarms	block_dcx deblock_dcx block_dcx deblock_dcx	dist_print_dcx dist_print_dcx dist_alarm_dcx dist_alarm_dcx
	Block/deblock update of object	block_dcx deblock_dcx	proc_update_dcx proc_update_dcx
	Acknowledge alarm	acknowledge_dcx	edum_ev_prop
digital output	Value	onopenstart_dcx offclose_dcx	output_dcx output_dcx
	Block/deblock alarms	block_dcx deblock_dcx block_dcx deblock_dcx	dist_print_dcx dist_print_dcx dist_alarm_dcx dist_alarm_dcx
	Block/deblock update of object	block_dcx deblock_dcx	output_dcx output_dcx
	Acknowledge alarm	acknowledge_dcx	edum_ev_prop

The external computer must answer a select order from an ABB Master Station. A select order from an ABB Master Station is identified by dest_ch_id, id, opcode and opprop.

opcode   = select_dcx
typ_of_req = 0; The user is an operator

An object selected from an ABB Master Station cannot be controlled by another user.

The ABB Master Station releases the object with a deselect order:

opcode   = deselect_dcx
typ_of_req = 0; The user is an operator

Signal sel_answer:

m	n	n+1	n+2	n+4	n+6	n+7	n+8	n+10	n+30	n+32	n+60	n+62	n+68	n+72
address part	mmi num	dummy	LF	LR	stat	sub_ system	length string	object name	length string	obj_descr	length string	obj_unit	a_value	d_value

- **stat:**

  grant = 0 => the object is selected
  alrd_sel = 1 => the object is already selected

- **a_value** (real)**:** current value for an analog object
- **d_value** (intl)**:** current value for a digital object

The select answer data part length is 76 bytes.

For Pascal layout, see Section 3.6.4.2 Pascal Layout Select Answer.

3.5.3.5 System Text

If an order or a TTD request is illegal, e.g., illegal limits, you receive a system text message from the object handler. The message includes an address for the object and an error code (text_index).

A system text message is identified by dest_ch_id and id.

id = syst_text

m	n	n+1	n+2	n+4	n+6	n+7	n+8	n+9	n+10
address part	mmi	dummy	LF	LR	type	netw	node	prop & logg	text_index

Only mmi number and text index are used in System Text signal.

- **mmi:** mmi number (1 or 2)
- **text index:** see Table 3-13

Syst_text data part length 12 bytes.

For Pascal layout, see Section 3.6.5.1 Pascal Layout syst_text.

3.5.4 Data Part Application Function

The application function to use via GCOM is the DSC function. It transfers data cyclically between the external computer and the ABB Master databases in the form of Data Set Packets (DSPs). External computers can also get DSPs from ABB Master on request.

3.5.4.1 Data Set Communication (DSC)

One DSP can consist of one to 24 values. A value can be one (1) real, one (1) integer or 32 digital values (boolean). The data layout of a DSP must be defined by the application programmer in the ABB MasterPiece or Advant Controller and in the external computer.

External computers can send and receive up to 255 DSPs to/from each ABB MasterPiece or Advant Controller.

Use the function DSC if the external computer continuously controls a process part. The advantage of DSC is that if data do not appear in a cycle time at the database in ABB Master, the application software (PC program) can interlock the process or continue by using default data. It is also possible to indicate to the operator in the ABB Master system that the object is handled by an external computer.

The cycle time for data transfer between ABB Master and an external computer is a parameter in the ABB MasterPiece or Advant Controller database.

A DSP message is identified by destination channel identity (dest_ch_id).

In a cyclic DSC message:

dest_ch_id = CXADSP

In a DSC request message:

dest_ch_id = CXADSD
resp_ch_id = CXADSP

Source network, source node and m_code (1 - 255) in a cyclic or requested DSP message to an external computer define the DSP. Destination node and m_code in a cyclic or on request DSP message from external computer define the DSP.

Signal identity (id) and dual computer are not a part of a DSP message.

DSC cyclic

The message is identified by dest_ch_id. Id and dual are not part of the address part (i.e., address part = 12 bytes).

M_code is used to define the DSP number (DSP ident) together with the address part.

m	n	n+4
address part	value 1	value 2 →

n+8	n+11	n+95
→ value 3		value 24

- **value**: intl

The size of the data part is 4 to 96 bytes.

DSC request

The external computer can request DSPs from the MasterPiece or Advant Controller. As a response to the request, the external computer receives a DSP message with the same format as the cyclic (address part = 12 bytes).

Id and dual are not part of the address part. M_code is used to define the DSP number (DSP ident) together with the address part.

m	n	n+1	n+2	n+3
address part	dsd	netw	node	stat

- **dsd** (intb) is the number on the requested DSP in the MasterPiece or Advant Controller.
- **netw** (intb) is the external computers own network number.
- **node** (intb) is the external computers own node number.
- **stat** (intb) is always equal to one (1).

The size of the data part is 4 bytes.

3.5.5 Data Part System Functions

The system functions are:

- **Symbolic name translation.** The external computer can send the symbolic name of an object to a MasterPiece or Advant Controller. The MasterPiece or Advant Controller translates the name to a logical database reference (LF and LR (refadr)) and returns it to the external computer. With this function, the external computer can build up a cross-reference table between logical database addresses in the ABB Master system and the external computer.
- **Clock synchronization.**This function synchronizes the system clocks in both systems.

The ABB Master system or one of the external computers connected to the GCOM subnet can handle the clock synchronization.

- **Communication status.** The function communication status (system status) enables the external computer to supervise units (directly) connected to an ABB Master unit. The request must be done cyclically.
- **Event handling.** The external computer can subscribe to system events from the ABB MasterGate (if it is connected to an MG 230).

The ABB Master Station units in the ABB Master system can use the same functions against the external computer (if the functions are implemented on the application layer with the same interface as in the ABB Master system).

3.5.5.1 Symbolic Name Translation

The external computer can send the symbolic name of an object to a Controller. The Controller translates the name to a logical database reference (LF and LR (refadr)) and returns it to the external computer. With this function, the external computer can build up a cross-reference table between logical database addresses in the ABB Master system and the external computer.

Symbolic name translation from external computer:

Symbolic name translation messages from the external computer (request) and answers for this are identified by destination channel identity (dest_ch_id). Id and dual are not a part of the address part (i.e., address part = 12 bytes).

Request:

dest_ch_id = CXNMC2
resp_ch_id = CXNMC1

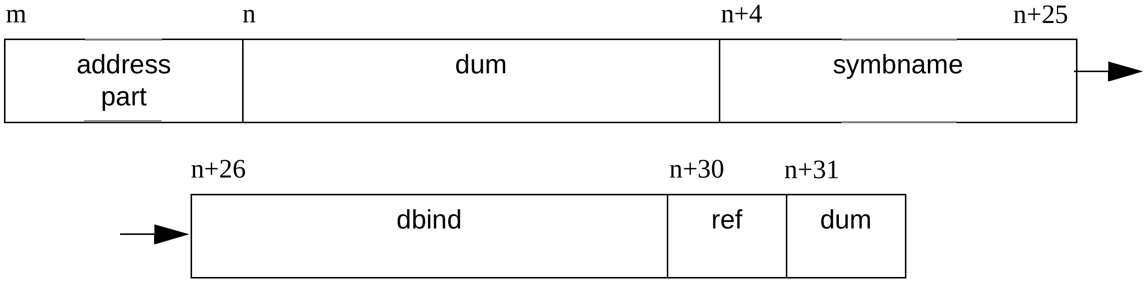
m	n	n+4
address part	dum	symbname

- **dum** (intl): free to use. Returned in the answer.
- **symbname** (string (.20.)): object name (maximum 20 characters).

The message’s data part length is 26 bytes.

Answer:

dest_ch_id = CXNMC1
repd_ch_id = CXNMC2



- **dum** (intl): returned from the request
- **symbname** (string (.20.)): object name
- **dbind** (intl): object reference (LF and LR)
- **ref** (intb): object type (reference type).

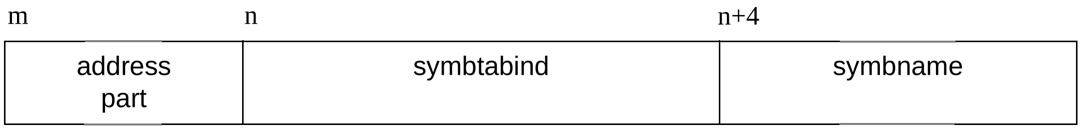
The message’s data part length is 32 bytes.

Symbolic name translation to external computer:

Symbolic name translation messages to external computers and answers for this are identified by destination channel identity (dest_ch_id). Id and dual are not a part of the address part (i.e., address part = 12 bytes).

Request:

dest_ch_id = CXNMC2
resp_ch_id = CXNMC1

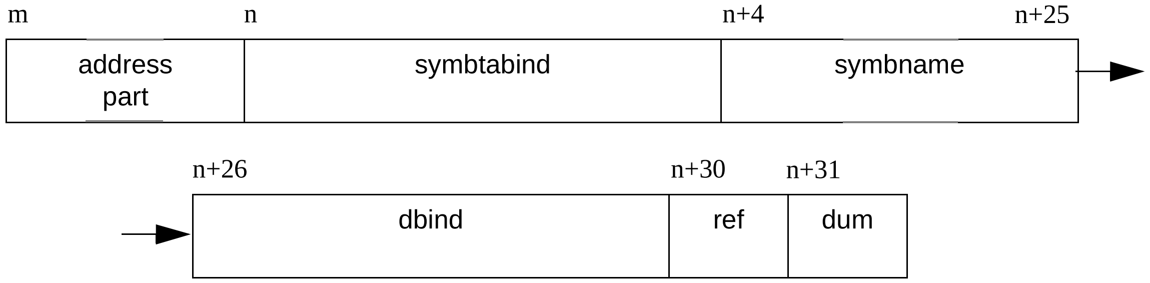


- **symbtabind** (intl): must be returned by the external computer.
- **symbname** (string (.20.)): object name (maximum 20 characters).

The message’s data part length is 26 bytes.

Answer:

dest_ch_id = resp_ch_id in the request
m_code = m_code in the request



- **symbtabind** (intw): returned from the request
- **symbname** (string (.20.)): object name (maximum 20 characters)
- **dbind** (intl): logical file (intw) and logical record (intw) for the object in the external computer
- **ref** (intb): object type (reference type)
- **dum** (intb): dummy.

The message's data part length is 32 bytes.

3.5.5.2 Clock Synchronization

This function synchronizes the system clocks in both systems.

The ABB Master system or one of the external computers connected to the GCOM subnet can handle the clock synchronization.

When the external computer starts, the ABB Master system sends a clock synchronization message and, thereafter, it cyclically sends a synchronization message to the external computer. If the external computer handles the clock synchronization, it sends the messages cyclically every 10th minute.

The message layout is the same in both directions.

The message is identified by dest_ch_id.

Clock synchronization message:

dest_ch_id = CXACSM
resp_ch_id = not used

The message is identified with dest_ch_id. Id and dual are not a part of the address part (address part = 12 bytes).

m	n	n+1	n+2	n+6
address part	mode	dum	external day	external time

- **mode** (intb): not used in messages to external computer. Always 0 in messages to ABB Master.
- **dum** (intb): not used
- **date** (intl): external day = number of days since 1 january 1980. Binary coded with 1 january 1980 as day 1 (n+2 is the most significant byte).
- **time** (intl): external time = number of 0,1 ms since midnight (n+6 is the most significant byte).

3.5.5.3 Communication Status

The function communication status (system status) enables the external computer to supervise units (directly) connected to an ABB Master unit. The request must be done cyclically.

Communication status request:

dest_ch_id = DCSC00
resp_ch_id = DCPC11 if mmi_1 and DCPD12 if mmi_2
id = sub_req

A request for communication status is made as a subscription request with a special logical file and reftype. The message is identified by dest_ch_id and id. It causes the addressed node to answer with the communication status to all its connected nodes. The answer is always demand type, irrespective of sub_type. The external computer addresses the message to the unit (end node) where the communication status is to be examined.

m	n	n+1	n+2	n+4	n+6	n+7	n+8	n+12	n+13
address part	mmi_ num	number	LF	LR	access_ type	sub_ type	send_condition	reftype	dummy

- **number** is the number of references in the request (always = 1 for communication status).

- **LF** should be set to the value 29344.
- **LR**, **access_type**, **sub_type** and **send_condition** must always be zero (not used by the external computer).
- **ref_type** must have the value 18.

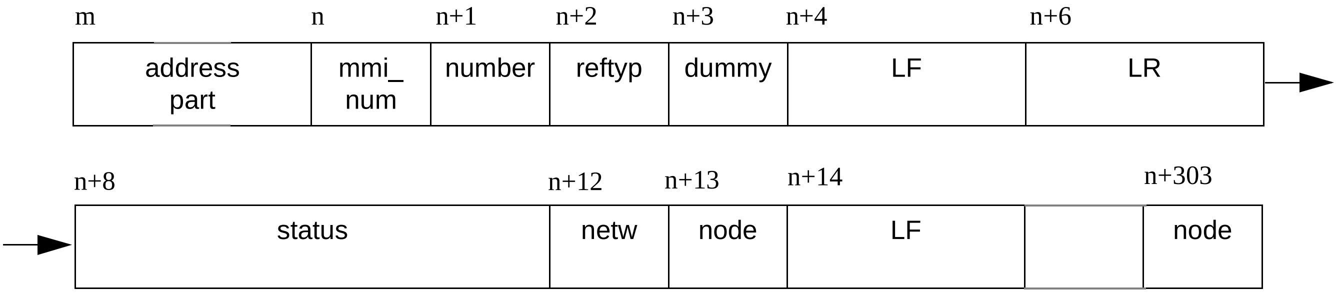
The message's data part length is 14 bytes.

For Pascal layout, see Section 3.6.6.1 Pascal Layout Communication Status.

Communication status response message:

dest_ch_id = resp_ch_id in request
resp_ch_id = not used
id = stat_deup

An answer on a communication status request can contain the status of from one to 30 connected nodes. The message is identified by dest_ch_id and id. For each node, the part starting with LF and ending with node is repeated. If the requested node has more than 30 connections, further signals are sent.



- **number** is the number of connection statuses in the message.
- **ref_type** must have the value 18.
- **LF** and **LR** contains no useful information.

- **Status** gives the communication status for the connection to the unit defined by **netw** and **node**. Read the bit map in status as follows:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 3-12. Bit map for status

bit no	description (when bit n = 1)
0	Link down
1	
2	MB200 bus A down
3	MB200 bus B down
4	MB200 Node busmaster
5	MB100 link
6	MB200 link
7	MB300 link
8	
9	CNC link
10	MVI link
11	GCOM link
12	RCOM link
13	
14	
15	

Bus A down and Bus B down are only valid if the node requested is busmaster on MB200.

Data part length is always 304 bytes

For Pascal layout, see Section 3.6.6.1 Pascal Layout Communication Status.

3.5.5.4 Event Handling

MG-event subscription

An external computer can subscribe to system events (not process events) from the ABB MasterGate (if it is connected to an MG 230). The function is located in the MasterGate node:

dest_netw = the control network to which the MG 230 is connected
dest_node = 127

Subscription request:

dest_ch_id = DCWC10
id = node_init

The m_code in the address part means:

m_code = 1; System event from MasterGate

The message has no data part.

The external computer gets the following response to the request:

dest_ch_id = not used
id = node_resp

The subscription response is the acknowledge on the subscription request.

m	n	n+1
address part	init	dum

- **init** (intb): true (1) => the request was successful. False(0) => the request failed.
- **dum** (intb): dummy.

System Event

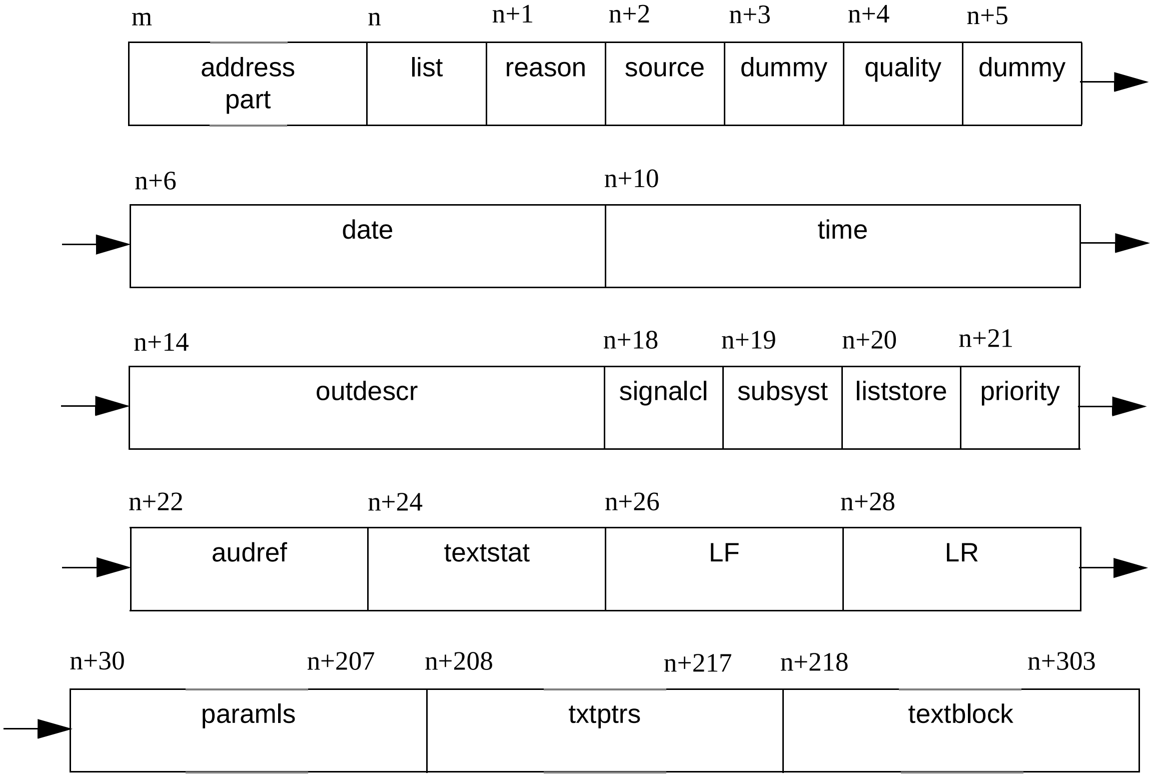
The system messages from the MasterGate are used to log the system events on a printer.

The text layout on the printer should be date, time and textblock. If the quality of date and time is bad (not equal to zero), it should not be present. The textblock is in ASCII and is printed out from an offset of 22.

The message is identified by dest_ch_id and signal identity (id).

dest_ch_id = DCCC40
id = sys_event

.



sys_event data part length is 304 bytes.

For Pascal layout, see Section 3.6.7.1 Pascal Layout System Events.

3.6 Configuration/Application Building Menus

In the product GCOM for VAX/VMS, the data types that correspond to the declarations in this chapter are included.

3.6.1 Data Subscription

3.6.1.1 Pascal Layout Subscription Request and Clear

SIGNAL: sub_req and sub_clear

```
dcxsubel=       RECORD (* subscription types *)
                access_type      :dcxaccesstype;
                sub_type         :dcxsubtype;
                send_condition : intl;
                END;

dcxreqinf =     RECORD
                refadr           :czdbind;
                subel            :dcxsubel;
                reftyp           :dcxreftype;
                dumsize          : intw
                END;

dcxxsubreq =    RECORD
                mmi_num          : dcxmminum;
                number           : intb;
                req_inf          : ARRAY(.1..max_subreq.) OF dcxreqinf;
                END;

dcxssubreq =    RECORD (* subscription request or clear *)
                header           : dcxheader;
                subreq           : dcxxsubreq;
                END;
```

3.6.1.2 Pascal Layout mmi clear

SIGNAL: mmi_clear

```
dcxxmmicl =     RECORD
                mmi_num          : dcxmminum;
                dumsize          : intw;
                END;

dcxsmmicl =     RECORD
                header           : dcxheader;
                mmicl            : dcxxmmicl;
                END;
```

3.6.2 Answer on Data Subscription

3.6.2.1 Pascal Layout for AI and AO Objects

SIGNAL: a_cupdate

```
dcxxxacup =   RECORD
              refadr        : czdbind;
              status        : intl;
              value         : real;
              nodec         : intb;
              dumsize       : intw;
              END;

dcxxacup =    RECORD
              mmi_num       : dcxmminum;
              number        : intb;
              typ           : dcxreftyp;
              inf           : ARRAY(.1..max_acup.) OF dcxxxacup;
              END;

dcxsacup =    RECORD  (* corresponds to access type ain_ab1_dcx and aout_ab1_dcx *)
              header        : dcxheader;
              acup          : dcxxacup;
              END;
```

SIGNAL: na_demand

```
dcxxxnad =    RECORD
              refadr        : czdbind;
              status:         intl;
              value         : real;
              nodec         : intb;
              range_max     : real;
              range_min     : real;
              name          : STRING (.20.);
              description   : STRING (.28.);
              dumsize       : intw;
              CASE integer OF
               1: (unit     : ARRAY(.1..6.) OF char);
               2: (intunit  : ARRAY(.1..6.) OF intb);
              END;

dcxxnad =     RECORD
              mmi_num       : dcxmminum;
              number        : intb;
              typ           : dcxreftyp;
              inf           : ARRAY(.1..max_nad.) OF dcxxxnad;
              END;
```

```
dcxsnad =     RECORD  (* corresponds to access type ain_ab2_dcx and aout_ab2_dcx *)
              header        : dcxheader;
              ad            : dcxxnad;
              END;
```

SIGNAL: la_demand

```
dcxxxlad =    RECORD
              refadr        : czdbind;
              status        : intl;
              value         : real;
              nodec         : intb;
              range_max     : real;
              range_min     : real;
              hilim2        : real;
              hilim1        : real;
              lolim1        : real;
              lolim2        : real;
              name          : STRING (.20.);
              description   : STRING (.28.);
              dumsize       : intw;
              CASE integer OF
               1 : (unit    : ARRAY(.1..6.) OF char)
               2 : (intunit : ARRAY(.1..6.) OF intb);
              END;

dcxxlad =     RECORD
              mmi_num       : dcxmminum;
              number        : intb;
              typ           : dcxreftyp;
              inf           : ARRAY(.1..max_lad.) OF dcxxxlad;
              END;

dcxslad =     RECORD  (* corresponds to access type ain_ab3_dcx *)
              header        : dcxheader;
              lad           : dcxxlad;
              END;
```

SIGNAL: aoDevDupdate

```
dcxxxaodev =   RECORD
               refadr          : czdbind;
               status          : intl;
               value           : real;
                nodec          : intb;
               range_max       : real;
               range_min       : real;
               name            : STRING (.20.);
               description     : STRING (.28.);
               subsystem       : intb;
               class           : intb;
               max_lim         : real;
               min_lim         : real;
               dumsize         : intw;
               CASE integer OF
               1 : (unit       : ARRAY(.1..6.) OF char);
                2 : (intunit   : ARRAY(.1..6.) OF intb);
               END;

dcxxaodev =    RECORD
               mmi_num         : dcxmminum;
               number          : intb;
               typ             : dcxreftyp;
               inf             : ARRAY(.1..maxAoDev.) OF dcxxxaodev;
               END;

dcxsaodev =    RECORD  (* corresponds to access type aout_ab3_dcx *)
               header          : dcxheader;
               aodev           : dcxxaodev;
               END;
```

SIGNAL: aiDevDupdate

```
dcxxxaodev =   RECORD
               refadr        : czdbind;
               status        : intl;
               value         : real;
               nodec         : intb;
               range_max     : real;
               range_min     : real;
               hilim2        : real;
               hilim1        : real;
               lolim1        : real;
               lolim 2       : real;
               name          : STRING (.20.);
               description   : STRING (.28.);
               subsystem     : intb;
               class         : intb;
               lim1_treat    : intw;
               lim2_treat    : intw;
               dumsize       : intw;
               CASE integer OF
               1 : (unit     : ARRAY(.1..6.) OF char);
               2 : (intunit  : ARRAY(.1..6.) OF intb);
               END;

dcxxaodev =    RECORD
               mmi_num       : dcxmminum;
               number        : intb;
               typ           : dcxreftyp;
               inf           : ARRAY(.1..maxAiDev.) OF dcxxxaidev;
               END;

dcxsaodev =    RECORD  (* corresponds to access type ain_ab4_dcx *)
               header        : dcxheader;
               aodev         : dcxxaidev;
               END;
```

3.6.2.2 Status for AI and AO Objects

DCP_STATUS_AI 1 SIGNAL_STATUS Analog input

STATUS	BITNO	DESCRIPTION
DCP_IMPLEMENTED_AI	0	IMPLEMENTED
DCP_ERROR_AI	1	ERROR
DCP_UPDATED_AI	2	UPDATED
DCP_UPD_BLK_AI	3	UPD_BLK
DCP_MAN_ENTRY_AI	4	MAN_ENTRY
DCP_SELECTED_AI	5	SELECTED
DCP_NOR_OBJ_TREAT_AI	6	NORMAL_OBJ_TREAT
DCP_H2_REP_FAIL_CTRL_AI	7	H2_REPEAT_FAIL_CONTROL
DCP_ABOVE_HI_LIM2_AI	8	ABOVE_HI_LIM2
DCP_ABOVE_HI_LIM1_AI	9	ABOVE_HI_LIM1
DCP_BELOW_LO_LIM1_AI	10	BELOW_LO_LIM1
DCP_BELOW_LO_LIM2_AI	11	BELOW_LO_LIM2
DCP_ALARM_UNACK_AI	12	ALARM_UNACK
DCP_H1_REP_FAIL_CTRL_AI	13	H1_REPEAT_FAIL_CONTROL
DCP_DISTURBANCE_AI	14	DISTURBANCE
DCP_OVERFLOW_AI	15	OVERFLOW
DCP_ALARM_BLK_AI	16	ALARM_BLK
DCP_AL_PERIOD_BLK_AI	17	ALARM_PERIOD_BLK
DCP_PRINT_BLK_AI	18	PRINT_BLK
DCP_L1_REP_FAIL_CTRL_AI	19	L1_R_FCL
DCP_LINKED_AI	20	LINKED
DCP_RELINK_AI	21	RELINK
DCP_NOERR_AT_OVF_AI	22	NOERR_AT_OVF
DCP_TESTED_AI	23	TESTED
DCP_ACC_ERR_AI	24	ACC_ERR
DCP_LOCK_AI	25	LOCK
DCP_L2_REP_FAIL_CTRL_AI	26	L2_REPEAT_FAIL_CONTROL
DCP_ER_REP_FAIL_CTRL_AI	27	ERROR_REPEAT_FAIL_CONTROL
DCP_REPEAT_FAIL_BLK_AI	28	REPEAT_FAIL_CONTROL
DCP_AI_ACTION_AI	29	AI_ACTION
DCP_AI_RETRY_AI	30	AI_RETRY
DCP_ERR_CTRL_AI	31	ERR_CTRL

DCP_STATUS_AO	1	SIGNAL_STATUS Analog output

STATUS	BITNO	DESCRIPTION
DCP_IMPLEMENTED_AO	0	IMPLEMENTED
DCP_ERROR_AO	1	ERROR
DCP_AO_SPARE_2_AO	2	AO_SPARE_2
DCP_OUTP_BLK_AO	3	OUTP_BLK
DCP_AO_SPARE_4_AO	4	AO_SPARE_4
DCP_SELECTED_AO	5	SELECTED
DCP_NOR_OBJ_TREAT_AO	6	NORMAL_OBJ_TREAT
DCP_MAN_MODE_AO	7	MAN_MODE
DCP_ON_MAX_LIM_AO	8	ON_MAX_LIM
DCP_AO_SPARE_9_AO	9	AO_SPARE_9
DCP_AO_SPARE_10_AO	10	AO_SPARE_10
DCP_ON_MIN_LIM_AO	11	ON_MIN_LIM
DCP_ALARM_UNACK_AO	12	ALARM_UNACK
DCP_AO_SPARE_13_AO	13	AO_SPARE_13
DCP_AO_SPARE_14_AO	14	AO_SPARE_14
DCP_AO_SPARE_15_AO	15	AO_SPARE_15
DCP_ALARM_BLK_AO	16	ALARM_BLK
DCP_AL_PERIOD_BLK_AO	17	ALARM_PERIOD_BLK
DCP_PRINT_BLK_AO	18	PRINT_BLK
DCP_OUTP_RESTART_AO	19	OUTP_RESTART
DCP_LOCK_AO	20	LOCK
DCP_USE_MAX_LIM_AO	21	USE_MAX_LIM
DCP_USE_MIN_LIM_AO	22	USE_MIN_LIM
DCP_TESTED_AO	23	TESTED
DCP_AO_SPARE_24_AO	24	AO_SPARE_24
DCP_AO_SPARE_25_AO	25	AO_SPARE_25
DCP_LINKED_AO	26	LINKED
DCP_AO_SPARE_27_AO	27	AO_SPARE_27
DCP_AO_SPARE_28_AO	28	AO_SPARE_28
DCP_AO_SPARE_29_AO	29	AO_SPARE_29
DCP_AO_SPARE_30_AO	30	AO_SPARE_30
DCP_AO_SPARE_31_AO	31	AO_SPARE_31

3.6.2.3 Pascal Layout for DI and DO Objects

SIGNAL: d_eupdate

```
dcxxxdeup =   RECORD
              refadr         : czdbind;
              status         : intl;
              dumsize        : intw
              END;

dcxxdeup =    RECORD
              mmi_num        : dcxmminum;
              number         : intb;
              typ            : dcxreftyp;
              inf            : ARRAY(.1..max_deup.) OF dcxxxdeup;
              END;

dcxsdeup =    RECORD  (* corresponds to access type din_ab1_dcx and dout_ab1_dcx *)
              header         : dcxheader;
              deup           : dcxxdeup;
              END;
```

SIGNAL: dDevDupdate

```
dcxxxddev =   RECORD
              refadr         : czdbind;
              status         : intl;
              name           : STRING (.20.);
              description    : STRING (.28.);
              subsystem      : intb;
              class          : intb;
              value_treat    : intw;
              dumsize        : intw;
              END;

dcxxddev =    RECORD
              mmi_num        : dcxmminum;
              number         : intb;
              typ            : dcxreftyp;
              inf            : ARRAY(.1..maxDDev.) OF dcxxxddev;
              END;

dcxsddev =    RECORD  (* corresponds to access type din_ab2_dcx and dout_ab2_dcx *)
              header         : dcxheader;
              ddev           : dcxxddev;
              END;
```

3.6.2.4 Status for DI and DO Objects

DCP_STATUS_DI	1	SIGNAL_STATUS Digital input

STATUS	BITNO	DESCRIPTION
DCP_IMPLEMENTED_DI	0	IMPLEMENTED
DCP_ERROR_DI	1	ERROR
DCP_UPDATED_DI	2	UPDATED
DCP_UPD_BLK_DI	3	UPD_BLK
DCP_MAN_ENTRY_DI	4	MAN_ENTRY
DCP_SELECTED_DI	5	SELECTED
DCP_NOR_OBJ_TREAT_DI	6	NORMAL_OBJ_TREAT
DCP_ERR_CTRL_DI	7	ERR_CTRL
DCP_VALUE_DI	8	VALUE
DCP_SEC_VALUE_DI	9	SEC_VALUE
DCP_NORM_POSN_DI	10	NORM_POSN
DCP_SEC_NORM_POSN_DI	11	SEC_NORM_POSN
DCP_ALARM_UNACK_DI	12	ALARM_UNACK
DCP_REP_FAIL_BLK_DI	13	REPEAT_FAIL_BLK
DCP_DISTURBANCE_DI	14	DISTURBANCE
DCP_CALC_VALUE_DI	15	CALC_VALUE
DCP_ALARM_BLK_DI	16	ALARM_BLK
DCP_AL_PERIOD_BLK_DI	17	ALARM_PERIOD_BLK
DCP_PRINT_BLK_DI	18	PRINT_BLK
DCP_REP_FAIL_CTRL_DI	19	REPEAT_FAIL_CTRL
DCP_LOCK_DI	20	LOCK
DCP_INVERTED_DI	21	INVERTED
DCP_DUAL_IND_DI	22	DUAL_IND
DCP_TESTED_DI	23	TESTED
DCP_DI_SPARE_24_DI	24	DI_SPARE_24
DCP_V9_CONTROL_DI	25	V9_CONTROL
DCP_LINKED_DI	26	LINKED
DCP_DI_RETRY_DI	27	DI_RETRY
DCP_DIC_IND_RED_DI	28	DIC_IND_RED
DCP_DI_ACTION_DI	29	DI_ACTION
DCP_DIC_IND_YELLOW_DI	30	DIC_IND_YELLOW
DCP_DIC_IND_GREEN_DI	31	DIC_IND_GREEN

DCP_STATUS_DO 1 SIGNAL_STATUS Digital input

STATUS	BITNO	DESCRIPTION
DCP_IMPLEMENTED_DO	0	IMPLEMENTED
DCP_ERROR_DO	1	ERROR
DCP_DO_SPARE_2_DO	2	DO_SPARE_2
DCP_OUTP_BLK_DO	3	OUTP_BLK
DCP_DO_SPARE_4_DO	4	DO_SPARE_4
DCP_SELECTED_DO	5	SELECTED
DCP_NOR_OBJ_TREAT_DO	6	NORMAL_OBJ_TREAT
DCP_MAN_MODE_DO	7	MAN_MODE
DCP_VALUE_DO	8	VALUE
DCP_DO_SPARE_9_DO	9	DO_SPARE_9
DCP_START_VALUE_DO	10	START_VALUE
DCP_DO_SPARE_11_DO	11	DO_SPARE_11
DCP_ALARM_UNACK_DO	12	ALARM_UNACK
DCP_DO_SPARE_13_DO	13	DO_SPARE_13
DCP_DO_SPARE_14_DO	14	DO_SPARE_14
DCP_DO_SPARE_15_DO	15	DO_SPARE_15
DCP_ALARM_BLK_DO	16	ALARM_BLK
DCP_AL_PERIOD_BLK_DO	17	ALARM_PERIOD_BLK
DCP_PRINT_BLK_DO	18	PRINT_BLK
DCP_OUTP_RESTART_DO	19	OUTP_RESTART
DCP_LOCK_DO	20	LOCK
DCP_INVERTED_DO	21	INVERTED
DCP_DO_SPARE_22_DO	22	DO_SPARE_22
DCP_TESTED_DO	23	TESTED
DCP_ORDER_TO_DO	24	ORDER_TO
DCP_ORDER_FROM_DO	25	ORDER_FROM
DCP_LINKED_DO	26	LINKED
DCP_DO_SPARE_27_DO	27	DO_SPARE_27
DCP_DO_SPARE_28_DO	28	DO_SPARE_28
DCP_DO_SPARE_29_DO	29	DO_SPARE_29
DCP_DO_SPARE_30_DO	30	DO_SPARE_30
DCP_DO_SPARE_31_DO	31	DO_SPARE_31

3.6.2.5 Pascal Layout for PIDCON, RATIO and MANSTN

```
param2Type =  RECORD
              CASE integer OF
              1 : (param2_pid  : ARRAY(.1..48.) OF intb);
              2 : (param2_real : ARRAY(.1..12.) OF real);
              END;

param6Type =  RECORD
              CASE integer OF
              1 : (param6_pid  : ARRAY(.1..32.) OF intb);
              2 : (param6_real : ARRAY(.1..8.) OF real);
              END;

param7Type =  RECORD
              CASE integer OF
              1 : (param7_pid  : ARRAY(.1..48.) OF intb);
              2 : (param7_real : ARRAY(.1..12.) OF real);
              END;

param8Type =  RECORD
              CASE integer OF
              1 : (param8_pid  : ARRAY(.1..8.) OF intb);
              2 : (param8_real : ARRAY(.1..2.) OF real);
              END;

param9Type =  RECORD
              CASE integer OF
              1 : (param9_pid  : ARRAY(.1..16.) OF intb);
              2 : (param9_real : ARRAY(.1..4.) OF real);
              END;

param10Type = RECORD
              CASE integer OF
              1 : (param10pid  : ARRAY(.1..8.) OF intb);
              2 : (param10real : ARRAY(.1..2.) OF real);
              END;

munitType =   RECORD
              CASE integer OF
              1 : (munit_pid   : ARRAY(.1..6.) OF char);
              2 : (munit_intb  : ARRAY(.1..6.) OF intb);
              END;

p1unitType =  RECORD
              CASE integer OF
              1 : (p1unit_pid  : ARRAY(.1..6.) OF char);
              2 : (p1unit_intb : ARRAY(.1..6.) OF intb);
              END;
```

```
p2unitType =   RECORD
               CASE integer OF
               1 : (p2unit_pid  : ARRAY(.1..6.) OF char);
               2 : (p2unit_intb : ARRAY(.1..6.) OF intb);
               END;
```

SIGNAL: pid_cupdate

```
dcxxxpcup =    RECORD
               refadr          : czdbind;
               status_1_pid    : intl;
               status_2_pid    : intl;
               status_3_pid    : intl;
               status_4_pid    : intl;
               param6          : param6Type;
               param8          : param8Type;
               param9          : param9Type;
               pout_pid        : real;
               param10         : param10Type;
               mno_of_dec_pid : intb;
               dumsize         : intw;
               END;
dcxxpcup =     RECORD
               mmi_num         : dcxmminum;
               number          : intb;
               typ             : dcxreftyp;
               inf             : ARRAY(.1..max_pidcup.) OF dcxxxpcup;
               END;


dcxspcup =     RECORD          (* corresponds to access type ctrl_ab1_dcx *)
               header          : dcxheader;
               pidcup          : dcxxpcup;
               END;
```

SIGNAL: pid_n_dupdate

```
dcxxxpndup =  RECORD
              refadr          : czdbind;
              status_1_pid    : intl;
              status_2_pid    : intl;
              status_3_pid    : intl;
              status_4_pid    : intl;
              param6          : param6Type;
              param8          : param8Type;
              param9          : param9Type;
              pout_pid        : real;
              param10         : param10Type;
              mno_of_dec_pid : intb;
              mrg_min_pid     : real;
              mrg_max_pid     : real;
              name_pid        : STRING (.20.);
              description_pid  : STRING (.28.);
              dumsize         : intw;
              END;

dcxxpndup =   RECORD
              mmi_num         : dcxmminum;
              number          : intb;
              typ             : dcxreftyp;
              inf             : ARRAY(.1..max_pidndup.) OF dcxxxpndup;
              END;

dcxspndup =   RECORD  (* corresponds to access type ctrl_ab2_dcx *)
              header          : dcxheader;
              pidndup         : dcxxpndup;
              END;
```

SIGNAL: pid_l_dupdate

```
dcxxxpldup =   RECORD
               refadr          : czdbind;
               status_1_pid    : intl;
               status_2_pid    : intl;
               status_3_pid    : intl;
               status_4_pid    : intl;
               param6          : param6Type;
               param8          : param8Type;
               param9          : param9Type;
               pout_pid        : real;
               param10         : param10Type;
               mno_of_dec_pid : intb;
               mrg_min_pid     : real;
               mrg_max_pid     : real;
               param7          : param7Type;
               name_pid        : STRING (.20.);
               description_pid  :STRING (.28.);
               munit           : munitType;
               dumsize         : intw;
               END;

dcxxpldup =    RECORD
               mmi_num         : dcxmminum;
               number          : intb;
               typ             : dcxreftyp;
               inf             : ARRAY(.1..max_pidldup.) OF dcxxxpldup;
               END;

dcxspldup =    RECORD  (* corresponds to access type ctrl_ab3_dcx *)
               header          : dcxheader;
               pidldup         : dcxxpldup;
               END;

rsparam2Type=  RECORD
               CASE integer OF
               1 : (param2_rs  : ARRAY(.1..24.) OF intb);
               2 : (par2_real  : ARRAY(.1..6.) OF real);
               END;

rsparam6Type=  RECORD
               CASE integer OF
               1 : (param6_rs  : ARRAY(.1..24.) OF intb);
               2 : (par6_real  : ARRAY(.1..6.) OF real);
               END;

rsparam7Type=  RECORD
               CASE integer OF
               1 : (param7_rs  : ARRAY(.1..32.) OF intb);
               2 : (par7_real  : ARRAY(.1..8.) OF real);
               END;
```

```
rsparam8Type= RECORD
              CASE integer OF
              1 : (param8_rs  : ARRAY(.1..8.) OF intb);
              2 : (par8_real  : ARRAY(.1..2.) OF real);
              END;

rsparam9Type= RECORD
              CASE integer OF
              1 : (param9_rs  : ARRAY(.1..4.) OF intb);
              2 : (par9_real  : ARRAY(.1..1.) OF real);
              END;

rsmunitType=  RECORD
              CASE integer OF
              1 : (munit_rs   : ARRAY(.1..6.) OF char);
              2 : (munit_intb : ARRAY(.1..6.) OF intb);
              END;

rsrrunitType= RECORD
              CASE integer OF
              1 : (rrunit_rs  : ARRAY(.1..6.) OF char);
              2 : (rrunit_intb : ARRAY(.1..6.) OF intb);
              END;

rspounitType= RECORD
              CASE integer OF
              1 : (pounit_rs  : ARRAY(.1..6.) OF char);
              2 : (pounit_intb: ARRAY(.1..6.) OF intb);
                              END;

rsp1unitType= RECORD
              CASE integer OF
              1 : (p1unit_rs  : ARRAY(.1..6.) OF char);
              2 : (p1unit_intb: ARRAY(.1..6.) OF intb);
              END;
```

SIGNAL: rs_cupdate

```
dcxxxrscup =  RECORD
              refadr          : czdbind;
              status_1_rs     : intl;
              status_2_rs     : intl;
              status_3_rs     : intl;
              status_4_rs     : intl;
              param6          : rsparam6Type;
              param8          : rsparam8Type;
              param9          : rsparam9Type;
              pout_rs         : real;
              mno_of_dec_rd: intb;
              rno_of_dec_rs  : intb;
              pono_of_dec_rs: intb;
              param2          : rsparam2Type;
              dumsize         : intw;
              END;

dcxxrscup =   RECORD
              mmi_num         : dcxmminum;
              number          : intb;
              typ             : dcxreftyp;
               inf            : ARRAY(.1..max_rscup.) OF dcxxxrscup;
              END;

dcxsrscup =   RECORD  (* corresponds to access type rs_ab1_dcx *)
              header          : dcxheader;
              rscup           : dcxxrscup;
              END;
```

SIGNAL: rs_n_dupdate

```
dcxxxrsndup =  RECORD
               refadr          : czdbind;
               status_1_rs     : intl;
               status_2_rs     : intl;
               status_3_rs     : intl;
               status_4_rs     : intl;
               param6          : rsparam6Type;
               param8          : rsparam8Type;
               param9          : rsparam9Type;
               pout_rs         : real;
               mno_of_dec_rs: intb;
               mrg_min_rs      : real;
               mrg_max_rs      : real;
               param2          : rsparam2Type;
               rno_of_dec_rs   : intb;
               pono_of_dec_rs: intb;
               name_rs         : STRING (.20.);
               description_rs  : STRING (.28.);
               dumsize         : intw;
               END;

dcxxrsndup =    RECORD
               mmi_num         : dcxmminum;
               number          : intb;
               typ             : dcxreftyp;
               inf             : ARRAY(.1..max_rsndup.) OF dcxxxrsndup;
               END;

dcxsrsndup =    RECORD  (* corresponds to access type rs_ab2_dcx *)
               header          : dcxheader;
               rsndup          : dcxxrsndup;
               END;
```

SIGNAL: rs_l_dupdate

```
dcxxxrsldup =  RECORD
               refadr          : czdbind;
               status_1_rs     : intl;
               status_2_rs     : intl;
               status_3_rs     : intl;
               status_4_rs     : intl;
               param6          : rsparam6Type;
               param8          : rsparam8Type;
               param9          : rsparam9Type;
               pout_rs         : real;
               mno_of_dec_rs : intb;
               mrg_min_rs       : real;
               mrg_max_rs      : real;
               param7          : rsparam7Type;
               param2          : rsparam2Type;
               rno_of_dec_rs   : intb;
               pono_of_dec_rs : intb;
               name_rs         : STRING (.20.);
               description_rs  : STRING (.28.);
               munit           : rsmunitType;
               rrunit          : rsrrunitType;
               pounit          : rspounitType;
               dumsize         : intw;
               END;

dcxxrsldup =   RECORD
               mmi_num         : dcxmminum;
               number          : intb;
               typ             : dcxreftyp;
               END;

dcxsrsldup =   RECORD  (* corresponds to access type rs_ab3_dcx *)
               header          : dcxheader;
               rsldup          : dcxxrsldup;
               END;

msparam2Type=RECORD
               CASE integer OF
               1 : (param2_ms : ARRAY(.1..16.) OF intb);
               2 : (par2_real : ARRAY(.1..4.) OF real);
               END;

msparam6Type=RECORD
               CASE integer OF
               1 : (param6_ms : ARRAY(.1..16.) OF intb);
               2 : (par6_real : ARRAY(.1..4.) OF real);
               END;
```

```
msparam7Type=RECORD
             CASE integer OF
             1 : (param7_ms   : ARRAY(.1..24.) OF intb);
             2 : (par7_real   : ARRAY(.1..6.) OF real);
             END;

msmunitType= RECORD
             CASE integer OF
             1 : (munit_ms    : ARRAY(.1..6.) OF char);
             2 : (munit_intb  : ARRAY(.1..6.) OF intb);
             END;

mspounitType= RECORD
             CASE integer OF
             1 : (pounit_ms   : ARRAY(.1..6.) OF char);
             2 : (pounit_intb : ARRAY(.1..6.) OF intb);
             END;

msp1unitType= RECORD
             CASE integer OF
             1 : (p1unit_ms : ARRAY(.1..6.) OF char);
             2 : (p1unit_intb : ARRAY(.1..6.) OF intb);
             END;
```

SIGNAL: ms_cupdate

```
dcxxxmscup = RECORD
             refadr          : czdbind;
             status_1_ms     : intl;
             status_2_ms     : intl;
             status_3_ms     : intl;
             status_4_ms     : intl;
             param6          : msparam6Type;
             pout_ms         : real;
             pres1f_ms       : real;
             mno_of_dec_ms   : intb;
             pono_of_dec_ms  : intb;
             dumsize         : intw;
             END;

dcxxmscup =  RECORD
             mmi_num         : dcxmminum;
             number          : intb;
             typ             : dcxreftyp;
             inf             : ARRAY(.1..max_mscup.) OF dcxxxmscup;
             END;

dcxsmscup =  RECORD  (* corresponds to access type ms_ab1_dcx *)
             header          : dcxheader;
             mscup           : dcxxmscup;
             END;
```

SIGNAL: ms_n_dupdate

```
dcxxxmsndup= RECORD
             refadr          : czdbind;
             status_1_ms     : intl;
             status_2_ms     : intl;
             status_3_ms     : intl;
             status_4_ms     : intl;
             param6          : msparam6Type;
             pout_ms         : real;
             pres1f_ms       : real;
             mno_of_dec_ms  : intb;
             mrg_min_ms      : real;
             mrg_max_ms      : real;
             pono_of_dec_ms : intb;
             name_ms         : STRING (.20.);
             description_ms : STRING (.28.);
             dumsize         :intw;
             END;

dcxxmsndup =  RECORD
             mmi_num         : dcxmminum;
             number          : intb;
             typ             : dcxreftyp;
             inf             : ARRAY(.1..max_msndup.) OF dcxxxmsndup;
             END;

dcxsmsndup =  RECORD  (* corresponds to access type ms_ab2_dcx *)
              header         : dcxheader;
             msndup          : dcxxmsndup;
             END;
```

SIGNAL: ms_l_dupdate

```
dcxxxmsldup = RECORD
              refadr          : czdbind;
              status_1_ms     : intl;
              status_2_ms     : intl;
              status_3_ms     : intl;
              status_4_ms     : intl;
              param6          : msparam6Type;
              pout_ms         : real;
              pres1f_ms       : real;
              mno_of_dec_ms : intb;
              mrg_min_ms      : real;
              mrg_max_ms      : real;
              pono_of_dec_ms : intb;
              porg_min_ms     : real;
              porg_max_ms     : real;
              param7          : msparam7Type;
              p1rg_min_ms     : real;
              p1rg_max_ms     : real;
              p1no_of_dec_ms : intb;
              name_ms         : STRING (.20.);
              description_ms : STRING (.28.);
              munit           : msmunitType;
              pounit          : mspounitType;
              p1unit          : msp1unitType;
              dumsize         : intw;
              END;

dcxxmsldup =  RECORD
              mmi_num         : dcxmminum;
              number          : intb;
              typ             : dcxreftyp;
              inf             : ARRAY(.1..max_msldup.) OF dcxxxmsldup;
              END;

dcxsmsldup = RECORD  (* corresponds to access type ms_ab3_dcx *)
              header          : dcxheader;
              msldup          : dcxxmsldup;
              END;
```

3.6.2.6 Parameters and Status for PIDCON, RATIO and MANSTN

Parameters for PIDCON

PARAM1	PARAM1 (FAULT_4A) IS A BLOCK ITEM. ITS CONTENT IS NOT DECLARED AS PROPERTIES.
PARAM2	CONTROL VALUES FROM MMC. (ARRAY ITEM)

MMI_SP	SETPOINT VALUE.
MMI_MAN_OUT	MANUAL OUTPUT VALUE.
MMI_GAIN	GAIN VALUE.
MMI_TI	INTEGRATION TIME CONSTANT.
MMI_TD	DERIVATION TIME CONSTANT.
MMI_TF	FILTER TIME CONSTANT.
SCALE_A	SCALING CONSTANT A, EVALUATED BY OBJECT HANDLER, USED FOR SCALING IN PC_ELEMENT.
SCALE_B	SCALING CONSTANT B, EVELUATED BY OBJECT HANDLER, USED FOR SCALING IN PC_ELEMENT.
MMI_OUT_LL	LOW LIMITATION VALUE FOR OUTPUT.
MMI_OUT_HL	HIGH LIMITATION VALUE FOR OUTPUT.
SPARE_PARAM2_01	
SPARE_PARAM2_02	

PARAM3	CONTROL CONDITIONS. (BLOCK ITEM)

MMI_MANF	CONTROL MODE SELECTION (CMS) ORDER.
MMI_MAN	CMS ORDER MAN.
MMI_AUTO	CMS ORDER AUTO.
MMI_INT_BLK	CONDITION ITEGRATION BLOCK.
MMI_DER_BLK	CONDITION DERIVATION BLOCK.
MMI_E1	CMS ORDER E1.
MMI_E2	CMS ORDER E2.
MMI_E3	CMS ORDER E3.

FROM PC

PARAM4 LIMITATIONS OF ANALOG SIGNALS IN PC_ELEMENT. (BLOCK ITEM)

OUT=LL	TRUE = OUTPUT HAS REACHED LOW LIMIT VALUE.
OUT=HL	TRUE = OUTPUT HAS REACHED HIGH LIMIT VALUE.
SP=LL	TRUE = SETPOINT HAS REACHED LOW LIMIT VALUE.
SP=HL	TRUE = SETPOINT HAS REACHED HIGH LIMIT VALUE.

PARAM5 PC_ELEMENT CONDITIONS TO MMC. (BLOCK ITEM)
LAST 16 BITS ARE NOT DECLARED AS PROPERTIES.

BAL	TRUE = CONTROL MODE BALANCE SELECTED.
MAN	TRUE = CONTROL MODE MAN SELECTED.
AUTO	TRUE = CONTROL MODE AUTO SELECTED.
E1	TRUE = CONTROL MODE E1 SELECTED.
E2	TRUE = CONTROL MODE E2 SELECTED.
E3	TRUE = CONTROL MODE E3 SELECTED.
LOCAL_OUT	TRUE = LOCAL CONTROL SELECTED.
CLAMP_OUT	TRUE = CLAMP CONTROL SELECTED.
EXT_OUT_LIMIT	TRUE = OUTPUT LIMITATION FROM EXTERNAL SOURCE ONLY.
EXT_GAIN_ENBL	TRUE = GAIN FROM EXTERNAL SOURCE ONLY.
EXT_TI_ENBL	TRUE = INTEGRATION TIME CONSTANTS FROM EXTERNAL SOURCE ONLY.
EXT_TD_ENBL	TRUE = DERIVATION TIME CONSTANT FROM EXTERNAL SOURCE ONLY.
EXT_TF_ENBL	TRUE = DERIVATION FILTER TIME CONSTANT FROM EXTERNAL SOURCE ONLY.
EXT_INT_BLK	TRUE = INTEGRATION PART IS BLOCKED BY EXTERNAL CONNECTION.
EXT_DER_BLK	TRUE = DERIVATION PART IS BLOCKED BY EXTERNAL CONNECTION.
SPARE_PARAM5_15	

PARAM6 CONTROLS VALUES FROM PC_ELEMENT TO MMC. (ARRAY ITEM)

MV	MEASURED VALUE.
AUTOSP	AUTO SETPOINT.
WSP	WORKING SETPOINT.
DEVIATION	DEVIATION = MV - WSP.
OUT	OUTPUT VALUE.
TS	SAMPLE TIME.
MAN_OUT	MANUAL OUTPUT.
SPARE_PARAM6_01	

PARAM7 DATA TO PC ELEMENT FROM MMC (BLOCK ITEM).

MMI_SPL	LIMITATION VALUE, SETPOINT LOW.
MMI_SPH	LIMITATION VALUE, SETPOINT HIGH.
MMI_MVL2	ALARM LIMIT VALUE, L2, FOR MEASURED VALUE.
MMI_MVL1	ALARM LIMIT VALUE, L1, FOR MEASURED VALUE.
MMI_MVH1	ALARM LIMIT VALUE, H1, FOR MEASURED VALUE.
MMI_MVH2	ALARM LIMIT VALUE, H2, FOR MEASURED VALUE.
MMI_MV_HYST	ALARM UNIT HYSTERESIS, MEASUREDVALUE.
MMI_DEVL	ALARM LIMIT VALUE, LOW, FOR DEVIATION VALUE.
MMI_DEVH	ALARM LIMIT VALUE, HIGH, FOR DEVIATION VALUE.
MMI_DEV_HYST	ALARM LIMIT HYSTERESIS, DEVIATION VALUE.
SPARE_PARAM7_01	
SPARE_PARAM7_02	

PARAM8 DATA FROM PC ELEMENT TO MMC (BLOCK ITEM).

EXT_OUT_LL	EXTERNAL LIMITATION VALUE, LOW, FOR OUTPUT.
EXT_OUT_HL	EXTERNAL LIMITATION VALUE, HIGH, FOR OUTPUT.

PARAM9 DATA FROM PC ELEMENT TO MMC (BLOCK ITEM).

EXT_GAIN	EXTERNAL GAIN VALE.
EXT_TI	EXTERNAL TI VALUE.
EXT_TD	EXTERNAL TD VALUE.
EXT_TF	EXTERNAL TF VALUE.

PARAM10 DATA FROM PC ELEMENT TO MMC (BLOCK ITEM).

PRES1	PRES1 VALUE.
PRES2	PRES2 VALUE.

Status for PIDCON

DCP_STATUS_1_PID	1	CONTAINS STATUS,DIST_CTRL2, ALARM_F1_BLK,ALARM_F1_PERIOD_BLK, ALARM_F2_BLK,ALARM_F2_PERIOD_BLK, PARAM4
DCP_STATUS_2_PID	2	CONTAINS TREATED_FAULT_STATUS, PARAM3
DCP_STATUS_3_PID	3	CONTAINS COMMAND,PARAM5
DCP_STATUS_4_PID	4	CONTAINS TYPE_F1_CTRL,TYPE_F2_CTRL, ALARM_F1_UNACK,ALARMF2_UNACK

STATUS_1	BITNO	DESCRIPTION
DCP_IMPLEMENTED_PID	0	IMPLEMENTED
DCP_SPARE_STA_01_PID	1	SPARE_STATUS_01
DCP_RUNNING_PID	2	RUNNING
DCP_ACTUATOR_PID	3	ACTUATOR
DCP_MVDIR_PID	4	MVDIR
DCP_SELECTED_PID	5	SELECTED
DCP_SPARE_STA_06_PID	6	SPARE_STATUS_06
DCP_TESTED_PID	7	TESTED
DCP_PANEL_CTRL_PID	8	PANEL_CTRL
DCP_ON_OFF_CTRL_PID	9	ON_OFF_CTRL
DCP_SPARE_STA_10_PID	10	SPARE_STATUS_10
DCP_ALARM_UNACK_PID	16	ALARM_UNACK
DCP_REP_FAIL_BLK_PID	17	REPEAT_FAIL_BLK
DCP_DIST_PID	18	DIST
DCP_ALARM_F1_BLK_PID	19	ALARM_F1_BLK
DCP_AL_F1_PE_BLK_PID	20	ALARM_F1_PERIOD_BLK
DCP_ALARM_F2_BLK_PID	21	ALARM_F2_BLK
DCP_AL_F2_PE_BLK_PID	22	ALARM_F2_PERIOD_BLK
DCP_OUT_EQ_LL_PID	24	OUT=LL
DCP_OUT_EQ_HL_PID	25	OUT=HL
DCP_SP_EQ_LL_PID	26	SP=LL
DCP_SP_EQ_HL_PID	27	SP=HL

STATUS_2	BITNO	DESCRIPTION
DCP_TS_MV<L1_PID	0	TS_MV<L1
DCP_TS_MV>H1_PID	1	TS_MV>H1
DCP_TS_DEV<L_PID	2	TS_DEV<L
DCP_TS_DEV>H_PID	3	TS_DEV>H
DCP_TS_MV<L2_PID	4	TS_MV<L2
DCP_TS_MV>H2_PID	5	TS_MV>H2
DCP_TS_LOCAL_FL_PID	6	TS_LOCAL_FL
DCP_TS_MAN_FL_PID	7	TS_MAN_FL
DCP_TS_AUTO_FL_PID	8	TS_AUTO_FL
DCP_TS_E1_FL_PID	9	TS_E1_FL
DCP_TS_E2_FL_PID	10	TS_E2_FL

DCP_TS_E3_FL_PID	11	TS_E3_FL
DCP_AI_ERROR_PID	12	AI_ERROR
DCP_AO_ERROR_PID	13	AO_ERROR
DCP_DCM_ERROR_PID	14	DCM_ERROR
DCP_PC_BLK_PID	15	PC_BLK
DCP_SERVUC_PID	16	SERVUC
DCP_HW_ERR_PID	17	HW_ERR
DCP_MMI_MANF_PID	24	MMI_MANF
DCP_MMI_MAN_PID	25	MMI_MAN
DCP_MMI_AUTO_PID	26	MMI_AUTO
DCP_MMI_INT_BLK_PID	27	MMI_INT_BLK
DCP_MMI_DER_BLK_PID	28	MMI_DER_BLK
DCP_MMI_E1_PID	29	MMI_E1
DCP_MMI_E2_PID	30	MMI_E2
DCP_MMI_E3_PID	31	MMI_E3

STATUS_3	BITNO	DESCRIPTION
DCP_REMOTE_PID	0	REMOTE
DCP_CENTRAL_PID	1	CENTRAL
DCP_LOCAL_PID	2	LOCAL
DCP_SERVICE_UNIT_PID	3	SERVICE_UNIT
DCP_BAL_PID	16	BAL
DCP_MAN_PID	17	MAN
DCP_AUTO_PID	18	AUTO
DCP_E1_PID	19	E1
DCP_E2_PID	20	E2
DCP_E3_PID	21	E3
DCP_LOCAL_OUT_PID	22	LOCAL_OUT
DCP_CLAMP_OUT_PID	23	CLAMP_OUT
DCP_EXT_OUT_LI_PID	24	EXT_OUT_LIMIT
DCP_EXT_GAIN_ENB_PID	25	EXT_GAIN_ENBL
DCP_EXT_TI_ENBL_PID	26	EXT_TI_ENBL
DCP_EXT_TD_ENBL_PID	27	EXT_TD_ENBL
DCP_EXT_TF_ENBL_PID	28	EXT_TF_ENBL
DCP_EXT_INT_BLK_PID	29	EXT_INT_BLK
DCP_EXT_DER_BLK_PID	30	EXT_DER_BLK
DCP_SPARE_PA5_15_PID	31	SPARE_PARAM5_15

STATUS_4	BITNO	DESCRIPTION
DCP_PRINT_F1_BLK_PID	2	PRINT_F1_BLK
DCP_RE_F1_FAIL_C_PID	3	REPEAT_F1_FAIL_CTRL
DCP_PRINT_F2_BLK_PID	10	PRINT_F2_BLK
DCP_RE_F2_FAIL_C_PID	11	REPEAT_F2_FAIL_CTRL
DCP_AU_MV<L1_PID	16	AU_MV<L1
DCP_AU_MV>H1_PID	17	AU_MV>H1
DCP_AU_DEV<L_PID	18	AU_DEV<L

DCP_AU_DEV>H_PID	19	U_DEV>H
DCP_AU_MV<L2_PID	24	AU_MV<L2
DCP_AU_MV>H2_PID	25	AU_MV>H2

Parameters for MANSTN

PARAM1 PARAM1 (FAULT_3A) IS A BLOCK ITEM.
ITS CONTENT IS NOT DECLARED AS PROPERTIES.

PARAM2 CONTROL VALUES FROM MMC. (ARRAY ITEM)

MMI_OUT_REF	MANUAL OUTPUT REFERENCE VALUE.
MMI_OUT_LL	LOW LIMITATION VALUE FOR OUTPUT
MMI_OUT_HL	HIGH LIMITATION VALUE FOR OUTPUT
SPARE_PARAM2_01	

PARAM3 CONTROL CONDITIONS FROM MMC. (BLOCK ITEM)

MMI_MAN	CMS ORDER MANUAL.
MMI_E1	CMS ORDER E1 FROM MMC.

FROM PC

PARAM4 LIMITATIONS OF ANALOG SIGNALS IN PC_ELEMENT. (BLOCK ITEM)

OUT=LL	TRUE = OUTPUT HAS REACHED LOW LIMIT VALUE.
OUT=HL	TRUE = OUTPUT HAS REACHED HIGH LIMIT VALUE.

PARAM5 PC_ELEMENT CONDITIONS TO MMC. (BLOCK ITEM)

TRACKB=1	TRUE = TRACKING FUNCTION TRACKB=1.
MAN	TRUE = CONTROL MODE MAN SELECTED.
E1	TRUE = CONTROL MODE E1 SELECTED.
SPARE_PARAM5_03	
SPARE_PARAM5_04	

PARAM6 CONTROLS VALUES FROM PC_ELEMENT TO MMC. (ARRAY ITEM)

MV	MEASURED VALUE TO MMC.
OUTREF	OUTPUT REFERENCE VALUE.
OUT	OUTPUT VALUE TO MMC.
TS	TS

PARAM7 DATA TO PC ELEMENT FROM MMC (BLOCK ITEM).

MMI_MVL2	ALARM LIMIT VALUE, L2, FOR MEASURED VALUE.
MMI_MVL1	ALARM LIMIT VALUE, L1, FOR MEASURED VALUE.
MMI_MVH1	ALARM LIMIT VALUE, H1, FOR MEASURED VALUE.
MMI_MVH2	ALARM LIMIT VALUE, H2, FOR MEASURED VALUE.
MMI_MV_HYST	ALARM UNIT HYSTERESIS, MEASURED VALUE.
SPARE_PARAM7_01	

Status for MANSTN

		INTEGER LONG
DCP_STATUS_1_MS	1	CONTAINS STATUS,DIST_CTRL2, ALARM_F1_BLK,ALARM_F1_PERIOD_BLK, ALARM_F2_BLK,ALARM_F2_PERIOD_BLK, PARAM4
DCP_STATUS_2_MS	2	CONTAINS TREATED_FAULT_STATUS, PARAM3
DCP_STATUS_3_MS	3	CONTAINS COMMAND,PARAM5
DCP_STATUS_4_MS	4	CONTAINS TYPE_F1_CTRL,TYPE_F2_CTRL, ALARM_F1_UNACK,ALARMF2_UNACK

STATUS_1	BITNO	DESCRIPTION
DCP_IMPLEMENTED_MS	0	IMPLEMENTED
DCP_SPARE_STA_01_MS	1	SPARE_STATUS_01
DCP_RUNNING_MS	2	RUNNING
DCP_ACTUATOR_MS	3	ACTUATOR
DCP_MVDIR_MS	4	MVDIR
DCP_SELECTED_MS	5	SELECTED
DCP_SPARE_STA_06_MS	6	SPARE_STATUS_06
DCP_TESTED_MS	7	TESTED
DCP_PANEL_CTRL_MS	8	PANEL_CTRL
DCP_ON_OFF_CTRL_MS	9	ON_OFF_CTRL
DCP_SPARE_STA_10_MS	10	SPARE_STATUS_10
DCP_ALARM_UNACK_MS	16	ALARM_UNACK
DCP_REP_FAIL_BLK_MS	17	REPEAT_FAIL_BLK
DCP_DIST_MS	18	DIST
DCP_ALARM_F1_BLK_MS	19	ALARM_F1_BLK
DCP_AL_F1_PE_BLK_MS	20	ALARM_F1_PERIOD_BLK
DCP_ALARM_F2_BLK_MS	21	ALARM_F2_BLK
DCP_AL_F2_PE_BLK_MS	22	ALARM_F2_PERIOD_BLK
DCP_OUT_EQ_LL_MS	24	OUT=LL
DCP_OUT_EQ_HL_MS	25	OUT=HL

STATUS_2	BITNO	DESCRIPTION
DCP_TS_MV<L1_MS	0	TS_MV<L1
DCP_TS_MV>H1_MS	1	TS_MV>H1
DCP_TS_MV<L2_MS	2	TS_MV<L2
DCP_TS_MV>H2_MS	3	TS_MV>H2
DCP_AI_ERROR_MS	4	AI_ERROR
DCP_AO_ERROR_MS	5	AO_ERROR
DCP_DCM_ERROR_MS	6	DCM_ERROR
DCP_PC_BLK_MS	7	PC_BLK
DCP_SERVUC_MS	8	SERVUC
DCP_HW_ERR_MS	9	HW_ERR

DCP_MMI_MAN_MS	24	MMI_MAN
DCP_MMI_E1_MS	25	MMI_E1

STATUS_3	BITNO	DESCRIPTION
DCP_REMOTE_MS	0	REMOTE
DCP_CENTRAL_MS	1	CENTRAL
DCP_LOCAL_MS	2	LOCAL
DCP_SERVICE_UNIT_MS	3	SERVICE_UNIT
DCP_TRACKB_MS	16	TRACK_B
DCP_MAN_MS	17	MAN
DCP_E1_MS	18	E1
DCP_SPARE_PA5_03_MS	19	SPARE_PARAM5_3
DCP_SPARE_PA5_04_MS	20	SPARE_PARAM5_4

STATUS_4	BITNO	DESCRIPTION
DCP_PRINT_F1_BLK_MS	2	PRINT_F1_BLK
DCP_RE_F1_FAIL_C_MS	3	REPEAT_F1_FAIL_CTRL
DCP_PRINT_F2_BLK_MS	10	PRINT_F2_BLK
DCP_RE_F2_FAIL_C_MS	11	REPEAT_F2_FAIL_CTRL
DCP_AU_MV<L1_MS	16	AU_MV<L1
DCP_AU_MV>H1_MS	17	AU_MV>H1
DCP_AU_MV<L2_MS	24	AU_MV<L2
DCP_AU_MV>H2_MS	25	AU_MV>H2

Parameters for RATIOSTN

PARAM1 PARAM1 (FAULT_3A) IS A BLOCK ITEM.
ITS CONTENT IS NOT DECLARED AS PROPERTIES.

PARAM2 CONTROL VALUES FROM MMC. (ARRAY ITEM)

MMI_RATIOREF	RATIO REFERENCE VALUE.
MMI_MAN_OUT	MANUAL OUTPUT VALUE.
MMI_BIAS	BIAS VALUE.
MMI_OUT_LL	LOW LIMITATION VALUE FOR OUTPUT FROM MMC.
MMI_OUT_HL	HIGH LIMITATION VALUE FOR OUTPUT FROM MMC.
SPARE_PARAM2_01	

PARAM3 CONTROL CONDITIONS FROM MMC. (BLOCK ITEM)

MMI_MAN	CMS ORDER MAN FROM MMC.
MMI_AUTO	CMS ORDER AUTO FORM MMC.
MMI_E1	CMS ORDER E1 FROM MMC.

FROM PC

PARAM4 LIMITATIONS OF ANALOG SIGNALS IN PC_ELEMENT. (BLOCK ITEM)

OUT=LL	TRUE = OUTPUT HAS REACHED LOW LIMIT VALUE.
OUT=HL	TRUE = OUTPUT HAS REACHED HIGH LIMIT VALUE.
RATIO=LL	TRUE = RATIOREF HAS REACHED LOW LIMIT VALUE.
RATIO=HL	TRUE = RATIOREF HAS REACHED HIGH LIMIT VALUE.

PARAM5 PC_ELEMENT CONDITIONS TO MMC. (BLOCK ITEM)

TRACK=1	TRUE = TRACKING FUNCTION TRACK=1
MAN	TRUE = CONTROL MODE MAN SELECTED.
AUTO	TRUE = CONTROL MODE AUTO SELECTED.
E1	TRUE = CONTROL MODE E1 SELECTED.
EXT_BIAS_ENBL	TRUE = EXTERNAL BIAS ENABLE
EXT_OUT_LIMIT	TRUE = OUTPUT LIMITATION FROM EXTERNAL SOURCE ONLY
SPARE_PARAM5_06	

PARAM6 CONTROLS VALUES FROM PC_ELEMENT TO MMC (ARRAY ITEM)

MV	MEASURED VALUE TO MMC.
RATIOREF	RATIO REFERENCE.
WRATIO	WORKING SETPOINT .
OUT	OUTPUT VALUE.
MAN_OUT	MANUAL OUTPUT REFERENCE VALUE
TS	TS

PARAM7 DATA TO PC ELEMENT FROM MMC BLOCK ITEM.

MMI_RATIOL	LIMITATION VALUE, RATIO LOW.
MMI_RATIOH	LIMITATION VALUE, RATIO HIGH.
MMI_MVL2	ALARM LIMIT VALUE, L2, FOR MEASURED VALUE.
MMI_MVL1	ALARM LIMIT VALUE, L1, FOR MEASURED VALUE.
MMI_MVH1	ALARM LIMIT VALUE, H1, FOR MEASURED VALUE.
MMI_MVH2	ALARM LIMIT VALUE, H2, FOR MEASURED VALUE.
MMI_MV_HYST	ALARM UNIT HYSTERESIS, MEASURED VALUE.
SPARE_PARAM7_01	

PARAM8 DATA FROM PC ELEMENT TO MMC BLOCK ITEM.

EXT_OUT_LL	EXTERNAL LIMITATION VALUE, LOW, FOR OUTPUT.
EXT_OUT_HL	EXTERNAL LIMITATION VALUE, HIGH, FOR OUTPUT.

PARAM9 DATA FROM PC ELEMENT TO MMC CONTAINING EXT_BIAS ONLY

Status for RATIOSTN

		INTEGER LONG
DCP_STATUS_1_RS	1	CONTAINS STATUS,DIST_CTRL2, ALARM_F1_BLK,ALARM_F1_PERIOD_BLK, ALARM_F2_BLK,ALARM_F2_PERIOD_BLK, PARAM4
DCP_STATUS_2_RS	2	CONTAINS TREATED_FAULT_STATUS, PARAM3
DCP_STATUS_3_RS	3	CONTAINS COMMAND,PARAM5
DCP_STATUS_4_RS 4		CONTAINS TYPE_F1_CTRL,TYPE_F2_CTRL, ALARM_F1_UNACK,ALARMF2_UNACK

STATUS_1	BITNO	DESCRIPTION
DCP_IMPLEMENTED_RS	0	IMPLEMENTED
DCP_SPARE_STA_01_RS	1	SPARE_STATUS_01
DCP_RUNNING_RS	2	RUNNING
DCP_SPARE_STA_03_RS	3	SPARE_STATUS
DCP_SPARE_STA_04_RS	4	SPARE_STATUS
DCP_SELECTED_RS	5	SELECTED
DCP_SPARE_STA_06_RS	6	SPARE_STATUS_06
DCP_TESTED_RS	7	TESTED
DCP_PANEL_CTRL_RS	8	PANEL_CTRL
DCP_SPARE_STA_09_RS	9	SPARE_STATUS_09
DCP_SPARE_STA_10_RS	10	SPARE_STATUS_10
DCP_ALARM_UNACK_RS	16	ALARM_UNACK
DCP_REP_FAIL_BLK_RS	17	REPEAT_FAIL_BLK
DCP_DIST_RS	18	DIST
DCP_ALARM_F1_BLK_RS	19	ALARM_F1_BLK
DCP_AL_F1_PE_BLK_RS	20	ALARM_F1_PERIOD_BLK
DCP_ALARM_F2_BLK_RS	21	ALARM_F2_BLK
DCP_AL_F2_PE_BLK_RS	22	ALARM_F2_PERIOD_BLK
DCP_OUT_EQ_LL_RS	24	OUT=LL
DCP_OUT_EQ_HL_RS	25	OUT=HL
DCP_RATIO_EQ_LL_RS	26	RATIO=LL
CP_RATIO_EQ_HL_RS	27	RATIO=HL

STATUS_2	BITNO	DESCRIPTION
DCP_TS_MV<L1_RS	0	TS_MV<L1
DCP_TS_MV>H1_RS	1	TS_MV>H1
DCP_TS_MV<L2_RS	2	TS_MV<L2
DCP_TS_MV>H2_RS	3	TS_MV>H2
DCP_AI_ERROR_RS	4	AI_ERROR
DCP_DCM_ERROR_RS	6	DCM_ERROR
DCP_PC_BLK_RS	7	PC_BLK
DCP_SERVUC_RS	8	SERVUC

DCP_HW_ERR_RS	9	HW_ERR
DCP_MMI_MAN_RS	24	MMI_MAN
DCP_MMI_AUTO_RS	25	MMI_AUTO
DCP_MMI_E1_RS	26	MMI_E1

STATUS_3	BITNO	DESCRIPTION
DCP_REMOTE_RS	0	REMOTE
DCP_CENTRAL_RS	1	CENTRAL
DCP_LOCAL_RS	2	LOCAL
DCP_SERVICE_UNIT_RS	3	SERVICE_UNIT
DCP_TRACKB_RS	16	TRACK_B
DCP_MAN_RS	17	MAN
DCP_AUTO_RS	18	AUTO
DCP_E1_RS	19	E1
DCP_EXT_BIAS_ENBL_RS	20	EXT_BIAS_ENBL
DCP_EXT_OUT_LI_RS	21	EXT_OUT_LIMIT
DCP_SPARE_PA5_06_RS	22	SPARE_PARAM5_06

STATUS_4	BITNO	DESCRIPTION
DCP_PRINT_F1_BLK_RS	2	PRINT_F1_BLK
DCP_RE_F1_FAIL_C_RS	3	REPEAT_F1_FAIL_CTRL
DCP_PRINT_F2_BLK_RS	10	PRINT_F2_BLK
DCP_RE_F2_FAIL_C_RS	11	REPEAT_F2_FAIL_CTRL
DCP_AU_MV<L1_RS	16	AU_MV<L1
DCP_AU_MV>H1_RS	17	AU_MV>H1
DCP_AU_MV<L2_RS	24	AU_MV<L2
DCP_AU_MV>H2_RS	25	AU_MV>H2

REAL	NO	DESCRIPTION
DCP_MV_RS	1	MV
DCP_RATIOREF_RS	2	RATIOREF
DCP_WRATIO_RS	3	WRATIO
DCP_OUT_RS	5	OUT
DCP_MAN_OUT_RS	6	MAN_OUT
DCP_TS_RS	7	TS
DCP_EXT_OUT_LL_RS	9	EXT_OUT_LL
DCP_EXT_OUT_HL_RS	10	EXT_OUT_HL
DCP_PARAM9_RS	11	PARAM9
DCP_POUT_RS	15	POUT
DCP_PRES1_RS	16	PRES1
DCP_MRANGE_MIN_RS	18	MRANGE_MIN
DCP_MRANGE_MAX_RS	19	MRANGE_MAX
DCP_MMI_RATIOL_RS	20	MMI_RATIOL
DCP_MMI_RATIOH_RS	21	MMI_RATIOH
DCP_MMI_MVL2_RS	22	MMI_MVL2

DCP_MMI_MVL1_RS	23	MMI_MVL1
DCP_MMI_MVH1_RS	24	MMI_MVH1
DCP_MMI_MVH2_RS	25	MMI_MVH2
DCP_SPARE_PA7_01_RS	26	SPARE_PA7
DCP_MMI_RATIOREF_RS	28	MMI_RATIO_REF
DCP_MMI_MAN_OUT_RS	29	MMI_MAN_OUT
DCP_MMI_BIAS_RS	30	MMI_BIAS
DCP_RRANGE_MIN_RS	31	RRANGE_MIN
DCP_RRANGE_MAX_RS	32	RRANGE_MAX
DCP_SPARE_PA2_01_RS	33	SPARE_PA2
DCP_MMI_OUT_LL_RS	34	MMI_OUT_LL
DCP_MMI_OUT_HL_RS	35	MMI_OUT_HL
DCP_DEADZ_RS	36	DEADZ
DCP_PORANGE_MIN_RS	37	PORANGE_MIN
DCP_PORANGE_MAX_RS	38	PORANGE_MAX
DCP_P1RANGE_MIN_RS	39	P1RANGE_MIN
DCP_P1RANGE_MAX_RS	40	P1RANGE_MAX
DCP_PERC_MIN_RS	41	PERC_MIN
DCP_PERC_MAX_RS	42	PERC_MAX

3.6.3 Historical Time Tagged Data (TTD)

3.6.3.1 Pascal Layout for Subscription

SIGNAL: ttd_request

```
dcxname =       ARRAY (.1..20.) OF char;

dcxtreatttd=    (mom_ttd,
                mean_ttd,
                max_ttd,
                min_ttd);

dcximode =      (dir_index,
                sea_name,
                sea_dbind);

dcxtimdat =     RECORD
                date           : intl;
                time           : intl
                END;

dcxgetvttd =    RECORD
                log_ref        : czdbind;
                variable_ref   : intw;
                start_time     : dcxtimdat;
                num_values     : intw;
                sub_type       : dcxsubtype;
                add_inf        : boolean;
                dumsize        : intw      (* DSX use *)
                END;
```

```
dcxxreqttd =    RECORD
                mmi_num    : dcxmminum;
                CASE opcode : dcxopcode OF
                    getttdval_dcx :
                        (getv :
                        RECORD
                            num_var : intb;
                            get_val : ARRAY (.1..max_rgvttd.) OF dcxgetvttd
                        END);
                    getttdinf_dcx  :
                        (geti :
                        RECORD
                            log_ref : czdbind;
                            CASE geti_mode : dcximode OF
                                dir_index :
                                    (dirind :
                                    RECORD
                                        variable_ref : intw;
                                        dumsize : intw  (* DSX use *)
                                    END);
                                sea_name :
                                    (sname    :
                                    RECORD
                                        log_interval : dcxtimdat;
                                        variable_ref : intw;
                                        var_name : dcxname;
                                        var_prop : intw;
                                        treatment : dcxtreatttd;
                                        dumsize : intw  (* DSX use *)
                                    END);
                                sea_dbind :
                                    (sdbind   :
                                    RECORD
                                        network : cznetw;
                                        node : cznode;
                                        instance : czdbind;
                                        var_prop : intw;
                                        treatment : dcxtreatttd;
                                        dumsize : intw  (* DSX use *)
                                    END);
                        END);
                END;

dcxsreqttd =    RECORD
                header          :dcxheader;
                request         :dcxxreqttd
                END;
```

3.6.3.2 Pascal Layout for Data Answer

SIGNAL: ttd_data

```
dcxttdtype =   intb;

dcxttdstat =   intb;

dcxiwtype =    RECORD
               iqual          : intw;
               ivalue         : intw
               END;

dcxibtype =    RECORD
               iqual          : intw;
               spare          : intb;
               ivalue         : intb
               END;

dcxttdval =    RECORD
               CASE dcxttdtype OF
                   retype     : (revalue     : real);
                   iltype     : (ilvalue     : intl);
                   iwtype     : (iwvalue     : dcxiwtype);
                   ibtype     : (ibvalue     : dcxibtype)
               END;

dcxdbuf =      RECORD
               CASE boolean OF
                   true       : (call         : dcxmmhbuf);
                   false      : (user         : ARRAY [1..max_datttd] OF dcxttdval)
               END;

dcxtimdat =    RECORD
               date     :   intl;
               time     :   intl
               END;

dcxname =      ARRAY (.1..20.) OF char;

dcxdescr =     ARRAY (.1..28.) OF char;

dcxunit =      ARRAY (.1..6.) OF char;

dcxtreatttd =  (mom_ttd,
               mean_ttd,
               max_ttd,
               min_ttd);
```

```
dcxinfottd =   RECORD
               var_name        : dcxname;
               var_descr       : dcxdescr;
               unit            : dcxunit;
               range_min       : real;
               range_max       : real;
               no_of_dec       : intb
               END;

dcxaddittd =   RECORD
               log_name        : dcxname;
               log_interval    : dcxtimdat;
               var_info        : dcxinfottd;
               ref_type        : dcxreftyp;
               var_prop        : intw
               END;

dcxxdatttd =   RECORD
               status          : dcxttdstat;
               log_ref         : czdbind;
               variable_ref    : intw;
               val_type        : dcxttdtype;
               treatment       : dcxtreatttd;
               last_package    : boolean;
               add_inf         : boolean;
               additional_inf  : dcxaddittd;
               num_values      : intw;
               values          : dcxdbuf
               END;

dcxsdatttd =   RECORD
               header          : dcxheader;
               mmi_num         : dcxmminum;
               start_time      : dcxtimdat;
               data            : dcxxdatttd;
               dumesig         : intw (*DMX-use*)
               END;
```

3.6.3.3 Pascal Layout Log Variable

SIGNAL: ttd_infdata

```
dcxttdtype =   intb;

dcxttdstat =   intb;

dcxname =      ARRAY (.1..20.) OF char;

dcxdescr =     ARRAY (.1..28.) OF char;

dcxunit =      ARRAY (.1..6.) OF char;
```

```
dcxtreatttd =   (mom_ttd,
                mean_ttd,
                max_ttd,
                min_ttd);

dcxtimdat =     RECORD
                date            : intl;
                time            : intl
                END;

dcxinfottd =    RECORD
                var_name        : dcxname;
                var_descr       : dcxdescr;
                unit            : dcxunit;
                range_min       : real;
                range_max       : real;
                no_of_dec       : intb
                END;

dcxchinfttd =   RECORD
                network         : cznetw;
                node            : cznodeno;
                ref_type        : dcxreftyp;
                instance        : czdbind;
                CASE primary : boolean   OF
                    true :
                        (prim :
                        RECORD
                                var_property   :   intw;
                                test1_property :   intw;
                                test2_property :   intw;
                                rmin_property  :   intw;
                                rmax_property  :   intw;
                                nodec_property :   intw;
                                unit_property  :   intw;
                                test1_logic  :   boolean;
                                test2_logic  :   boolean
                        END);
                    false  :
                        (dep :
                        RECORD
                                var_index  :   intw;
                                var_property   :   intw
                        END)

                END;
```

```
dcxxinfttd =   RECORD
               status          : dcxttdstat;
               log_ref         : czdbind;
               log_name        : dcxname;
               log_interval    : dcxtimdat;
               variable_ref    : intw;
               var_info        : dcxinfottd;
               ref_info        : dcxchinfttd;
               val_type        : dcxttdtype;
               treatment       : dcxtreatttd;
               bad_val_lim     : intb;
               scaling         : boolean;
               scaling_c       : real;
               END;

dcxsinfttd =   RECORD
               header          : dcxheader;
               mmi_num         : dcxmminum;
               infdata         : dcxxinfttd
               END;
```

3.6.4 Order

3.6.4.1 Pascal Layout Order

SIGNAL: order

```
dcxxorder =    RECORD
               mmi_num         : dcxmminum;
               pres_func_id    : intb;
               refadr          : czdbind;
               opcode          : dcxopcode;
               opprop          : intw; (* Operation or event property *)
               typ_of_req      : dcxtypreq;
               event_pr        : boolean;
               a_value         : real;
               d_value         : intl;
               END;

dcxsorder =    RECORD  (* order to e.g. object-handler *)
               header          : dcxheader;
               inf             : dcxxorder;
               END;
```

3.6.4.2 Pascal Layout Select Answer

```
dcxselstat =    (* select answer status *)
                (grant,
                alrd_sel,
                not_allowed);

dcxobtxt =      RECORD (* object description texts *)
                objectname     : STRING(.20.);
                obj_descr      : STRING(.28.);
                obj_unit       : STRING(.6.); END;
```

SIGNAL: sel_answer

```
dcxxselans =    RECORD
                mmi_num        : dcxmminum;
                refadr         : czdbind;
                stat           : dcxselstat;
                sub_system     : intb;
                texts          : dcxobtxt;
                a_value        : real;
                d_value        : intl;
                END;

dcxsselans =    RECORD  (* selection answer *)
                header : dcxheader;
                selans : dcxxselans;
                END;
```

3.6.5 System Text

3.6.5.1 Pascal Layout syst_text

SIGNAL: syst_text

```
dcxref =        RECORD
                adr            : czdbind;
                typ            : dcxreftyp;
                network        : cznetw;
                node           : cznodeno;
                CASE intb OF
                    0 : (prop_no : intb) ; (* for access of flat structure *)
                    1 : (logg_variable_name : intb) ; (* for TTD *)
                END

dcxxsytext =    RECORD
                mmi_num        : dcxmminum;
                ref            : dcxref;
                text_index     : intw;
                END;
```

```
dcxssytext =    RECORD  (* system text presentation request *)
                header        : dcxheader;
                sytext        : dcxxsytext;
                END;
```

3.6.5.2 Text Pointers

Table 3-13. Text Pointers

text index/ instance no	reason (text)
4	OBJECT ALREADY SELECTED FROM ANOTHER WORK STATION
5	ILLEGAL OPERATION
13	NAME ALREADY EXIST
18	OBJECT NOT FOUND
19	RESOURCE ALLOCATED BY OTHER USER
40	ILLEGAL NAME OR NR
44	HIGH LIMIT 1
45	HIGH LIMIT 2
46	LOW LIMIT 1
47	LOW LIMIT 2
51	INVALID OPERATION CODE. CALL FOR SYSTEM MANAGER !
52	INVALID OPERATION PROPERTY. CALL FOR SYSTEM MANAGER !
53	OBJECT NOT SELECTED (OR NOT IMPLEMENTED).
54	OPERATION IS NOT ALLOWED WHEN PROCESS UPDATE IS NOT BLOCKED
55	LIMIT OUT OF RANGE. INPUT IGNORED
56	VALUE OUT OF RANGE. INPUT IGNORED
57	LIMIT NOT USED. INPUT IGNORED
58	MANUAL ORDERS BLOCKED. ORDER IGNORED
59	BLOCKING OF INTERGRATION NOT ALLOWED. ORDER IGNORED
60	BLOCKING OF DERIVATION NOT ALLOWED. ORDER IGNORED
61	MANUAL MODE SELECTION NOT ALLOWED
62	AUTO MODE SELECTION NOT ALLOWED
63	E1 MODE SELECTION NOT ALLOWED
64	E2 MODE SELECTION NOT ALLOWED
65	E3 MODE SELECTION NOT ALLOWED
66	SETPOINT IS TRACKING
67	OUTPUT VALUE CAN ONLY BE CHANGED IN MANUAL MODE
68	OUTPUT VALUE IS CONNECTED TO EXTERNAL REFERENCE
69	THREE POINT CONTROL NOT IMPLEMENTED. DZ NOT USED
70	M1 MAIN ORDER BLOCKED. ORDER IGNORED

Table 3-13. Text Pointers

text index/ instance no	**reason (text)**
71	MO MAIN ORDER BLOCKED. ORDER IGNORED
72	MO SPARE ORDER BLOCKED. ORDER IGNORED
73	M1 SPARE ORDER BLOCKED. ORDER IGNORED
74	OBJECT NOT SELECTED. ORDER IGNORED
75	DCM-COMMUNICATION FAILURE. ORDER IGNORED
76	ORDER NOT ALLOWED FOR DISTRIBUTED
77	OUTPUT BLOCKED. ORDER IGNORED
78	ORDER NOT ALLOWED IN THIS MODE. ORDER IGNORED
79	OBJECT ALREADY BLOCKED
80	OBJECT NOT BLOCKED
81	OBJECT ALREADY IN MANUAL MODE
82	OBJECT ALREADY IN AUTO MODE
83	BOARD ALREADY IN SERVICE
84	BOARD ALREADY OUT OF SERVICE
85	NO INSTANCES CONNECTED TO BOARD
86	RATIO CONNECTED TO EXTERNAL REFERENCE
87	MAIN OPEN ORDER BLOCKED. ORDER IGNORED
88	MAIN CLOSE ORDER BLOCKED. ORDER IGNORED
89	MAIN STOP ORDER BLOCKED. ORDER IGNORED
90	INCREASE SMALL STEP BLOCKED. ORDER IGNORED
91	INCREASE LARGE STEP BLOCKED. ORDER IGNORED
92	DECREASE SMALL STEP BLOCKED ORDER IGNORED
93	DECREASE LARGE STEP BLOCKED. ORDER IGNORED
94	SPARE OPEN ORDER BLOCKED. ORDER IGNORED
95	SPARE CLOSE ORDER BLOCKED. ORDER IGNORED
96	SPARE STOP ORDER BLOCKED. ORDER IGNORED
97	OBJECT ALREADY IN HOLD MODE
98	OBJECT NOT IMPLEMENTED
99	STEP CONDITIONS NOT FULFILLED
100	INPUT BLOCKED. STATUS CHECK NOT PERFORMED
101	INSTANCE DOES NOT EXIST OR BOARD NOT IMPLEMENTED
102	BOARD OUT OF SERVICE
103	HARDWARE ERROR REPORTED
104	OUTPUT TEMPORARY LOCKED. TRY AGAIN
105	OUTPUT FAILED
106	ATTEMPT TO READ INPUT FAILED
107	ILLEGAL DATA TYPE. CALL FOR SYSTEMS MANAGER
108	CONCEPT/PROPERTY NOT FOUND IN LF 1. CALL FOR SYSTEMS MANAGER
109	THE SEQUENCE IS NOT RUNNING. ORDER IGNORED
110	EVENT PRINT ALREADY BLOCKED

Table 3-13. Text Pointers

text index/ instance no	reason (text)
111	EVENT PRINT ALREADY DEBLOCKED
112	EVENT PROCESS ALREADY BLOCKED
113	EVENT PROCESS ALREADY DEBLOCKED
114	LIMIT IS NOT ACTIVE FOR THIS OBJECT
115	THE INPUT IS NOT CALCULATED. ORDER IGNORED
116	ATTEMPT TO CHANGE MAX LIMIT TO BE LESS THAN MIN LIMIT
117	ATTEMPT TO CHANGE MIN LIMIT TO BE GREATER THAN MAX LIMIT
118	FLAG IN MORD ALLOWED IS NOT TRUE. ORDER IS NOT PERFORMED.
119	FLAG IN VALUE ALLOWED IS NOT TRUE. ORDER IS NOT PERFORMED.
120	OBJECT IS SELECTED BY ANOTHER OPERATOR. ORDER IS NOT PERFORMED
121	VALUE OUT OF RANGE
122	ATTEMPT TO CHANGE OP_MAX TO BE < OP_MIN
123	ATTEMPT TO CHANGE OP_MIN TO BE > OP_MAX
124	ATTEMPT TO CHANGE SP_MAX < SP_MIN OR OUT OF RANGE OF MV-LIMITS
125	ATTEMPT TO CHANGE SP_MIN > SP_MAX OR OUT OF RANGE OF MV-LIMITS
150	TTD: A BACKUP IS BEING TAKEN FOR THE LOG
155	TTD: ILLEGAL MM NAME
161	TTD: ILLEGAL DATA TYPE FOR REFERENCED PROPERTY
175	TO MANY SUBSCRIPTIONS
251	TTD: ILLEGAL OPERATION CODE
252	TTD: LOG NOT FOUND
253	TTD: LOG NOT IMPLEMENTED
254	TTD: VARIABLE NOT FOUND
255	TTD: VARIABLE NOT IMPLEMENTED
256	TTD: VARIABLES OUT OF RANGE
257	TTD: NUMBER OF REQUESTED VALUES OUT OF RANGE
258	TTD: TIME INTERVAL COMPLETLY OUT OF RANGE
259	TTD: TIME INTERVAL OUT OF RANGE, OLDEST LIMIT
260	TTD: TIME INTERVAL OUT OF RANGE, YOUNGEST LIMIT
261	TTD: TIME INTERVAL OUT OF RANGE, OLDEST AND YOUNGEST LIMIT
262	TTD: LOGGING NOT ACTIVE
263	TTD: LOG ALREADY INACTIVE
264	TTD: LOGGING ALREADY ACTIVE
268	TTD: ILLEGAL LOGICAL FILE NUMBER
269	TTD: ILLEGAL SUBSCRIPTION CODE (SUB TYPE)
270	TTD: ILLEGAL DATA TYPE(TTD TYPE)

Table 3-13. Text Pointers

text index/ instance no	reason (text)
271	TTD: ILLEGAL VALUE IN TIME SPECIFICATION
272	TTD: THE LOG IS NO USER DATA LOG
273	TTD: ILLEGAL REFERENCE TYPE (REF TYPE)
274	TTD: ILLEGAL PROPERTY NUMBER
275	TTD: ADDITIONAL INFORMATION (ADD INF) MISSING
276	TTD: THE VARIABLE IS DEPENDENTLY LOGGED
277	TTD: REFERENCED CONCEPT NOT DEFINED IN THE NODE
278	TTD: REFERENCED INSTANCE NOT FOUND IN THE NODE
279	TTD: REFERENCED OBJECT OR LOG IS NOT IMPLEMENTED
280	TTD: REFERENCED VARIABLE IS NOT FOUND IN THE LOG
281	TTD: REFERENCED VARIABLE IS NOT IMPLEMENTED
282	TTD: REFERENCED VARIABLE IS NOT INITIATE
283	TTD: ACCESS TO PRIMARY LOGGED OBJECT FAILED
284	TTD: ILLEGAL PROPERTY REFERENCE FOR PRIMARY
287	TTD: PROGRAM ERROR
288	TTD: ILLEGAL LOG REFERENCE
289	TTD: THE SAMPLE INTERVALS DO NOT MATCH
290	TTD: REFERENCED LOG IS A USER DATA LOG
291	TTD: REFERENCED LOG IS A BACKUP LOG
292	TTD: ILLEGAL CODE FOR REQUESTING INFORMATION (GETI MODE)
293	TTD: LOG OR LOG INTERVAL MUST BE SPECIFIED
299	TTD: ILLEGAL LOGICAL FILE NUMBER
301	UNKNOWN OBJECT. ORDER IGNORED
306	UNKNOWN OBJECT. ORDER IGNORED
307	TEMPORARY OVERLOAD. PLEASE TRY AGAIN

3.6.6 Communication Status

3.6.6.1 Pascal Layout Communication Status

SIGNAL: sub_req

```
dcxsubel  =     RECORD (* subscription types *)
                access_type     : dcxaccesstype;
                sub_type        : dcxsubtype;
                send_condition : intl;
                END;

dcxreqinf =     RECORD
                refadr          : czdbind;
                subel           : dcxsubel;
                reftyp          : dcxreftype;
                dumsize         : intw
                END;

dcxxsubreq =    RECORD
                mmi_num         : dcxmminum;
                number          : intb;
                req_inf         : ARRAY(.1..max_subreq.) OF dcxreqinf;
                END;

dcxssubreq =    RECORD  (* subscription request *)
                header          : dcxheader;
                subreq          : dcxxsubreq;
                END;
```

SIGNAL: stat_deup

```
dcxwdeup   =    RECORD
                refadr          : czdbind;
                status          : intl;
                netw            : intb;
                node            : intb;
                END;

dcxxwdeup  =    RECORD
                mmi_num         : dcx_mminum;
                number          : intb;
                typ             : dcxreftyp;
                inf             : ARRAY (.1..dcwdmdeup.) OF dcxwdeup;
                END;

dcxswdeup  =    RECORD
                mheader         : czmsgheader;
                id              : dcxsigid;
                deup            : dcxxwdeup;
                END;
```

3.6.7 Event Handling

3.6.7.1 Pascal Layout System Events

SIGNAL: sys_event

```
dcxsysev =    RECORD (* system event ( formerly EV2 )*)
              header        : dcxheader;
              list          : dcxlstype;
              reason        : dcxevrea;
              source        : dcxsource;
              evtime        : dcxtime;
              outdescr      : intl;
              signalcl      : intb;
              subsyst       : intb;
              liststore     : boolean;
              priority      : dcxalpri;
              audref        : intw;
              textstat      : intw;
              textref       : czdbind;
              paramls       : dcxparls;
              txtptrs       : ARRAY(.1..dcx_noptrs.) OF intb;
              textblock     : ARRAY(.1..dcx_blksz.) OF char;
              END;
```

Chapter 4 Runtime Operation

4.1 Product Operation

Not applicable. Runtime operation is determined by your application in the external computer.

4.2 Operating Overview

Not applicable.

4.3 Runtime Tutorial

Not applicable.

4.4 Operating Instructions

Not applicable.

4.5 Runtime Operation Menus

Not applicable.

Chapter 5 Maintenance

5.1 Preventive Maintenance

If GCOM is installed in an AC400 or a MP 200/1, the GCOM subnet is part of the Control network to which the AC400/MP 200/1 belongs. For maintenance, see *MasterNet User's Guide*.

If GCOM is installed in a MG 230/1, the GCOM subnet is part of the Plant network. For maintenance, see *MasterGate 230/1 Installation and Error Codes*.

5.2 Hardware Indicators

See Section 5.1 Preventive Maintenance.

5.3 Error Messages

Section 5.1 Preventive Maintenance.

5.4 Fault Finding and User Repair

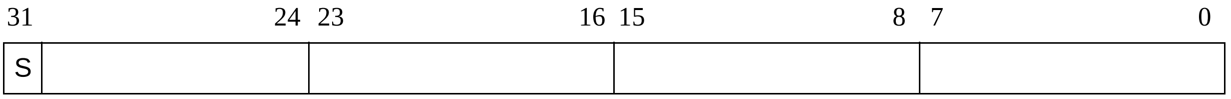
Section 5.1 Preventive Maintenance.

Appendix A Data Representation

A.1 Internal Representation

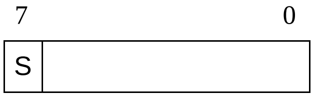
The floating point data in this section is generally valid for both standard floating point and fast floating point. Exceptions are noted.

A.1.1 Integer



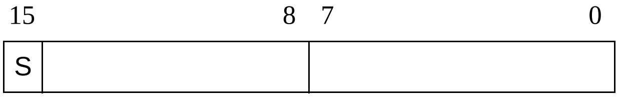
Size:	4 bytes; intl
Format:	Signed two's-complement
Range:	-2,147,483,648 to 2,147,483,647

For an integer subrange type within the range -128 to 127, inclusive:



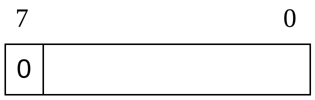
Size:	1 byte; intb
Format:	Signed two's-complement
Range:	-128 to 127

For an integer subrange type that extends outside the range -128 to 127 inclusive, but is within the range -32,768 to 32,767, inclusive.



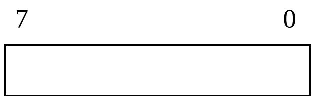
Size:	2 bytes; intw
Format:	Signed two's-complement
Range:	-32,768 to 32,767

A.1.2 Character



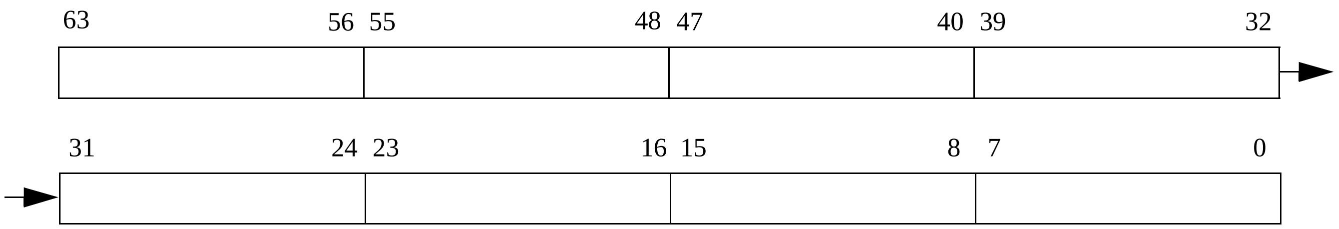
Size:	1 byte
Format:	7-bit ASCII
Range:	0 to 127

A.1.3 Boolean



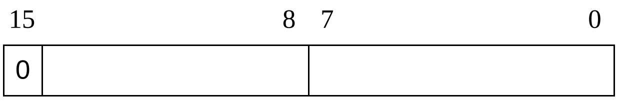
Size:	1 byte
Values:	0 = False 1 = True

A.1.4 Set



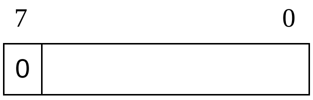
Size:	8 bytes
Range:	Up to 64 elements

A.1.5 Enumerated scalar

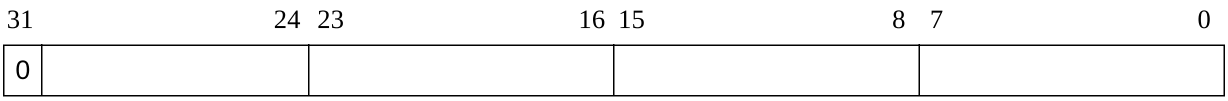


Size 2 bytes
Representation: 0 to 32, 767

:

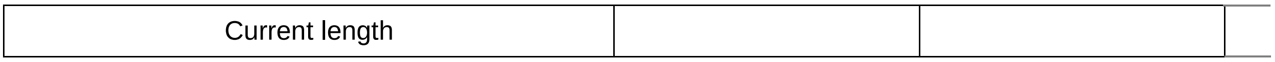


Size: 1 byte
Representation: 0 to 127



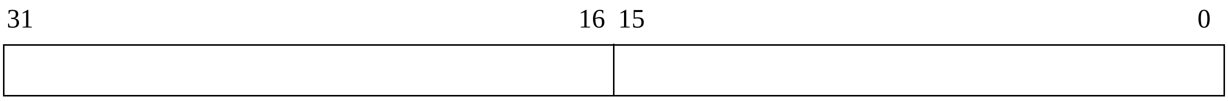
Size: 4 bytes
Representation: 0 to 2, 147, 483, 647

A.1.6 String



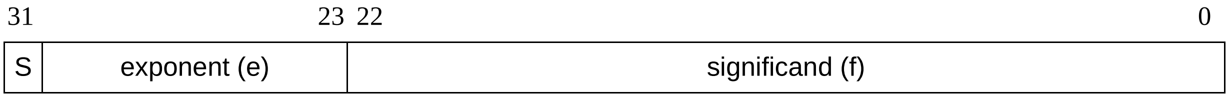
Size: 2 to 32766 bytes
Representation: Current length word and 0 to 32764 ASCII characters

A.1.7 Pointer



Size: 4 bytes
Range: 0 to 2^{32}-1

A.1.8 Real



Length in bits	32
Interpretation of sign:	
positive	0
negative	1
Normalized Numbers:	
interpretation of e	unsigned integer
bias of e	127
range of e	$0 < e < 255$
interpretation of significand	$1+f/(2^{23}-1)$
relation to representation of real numbers	$(-1)\ s \times 2^{(e-127)} \times 1+f/(2^{23}-1)$

NOTE

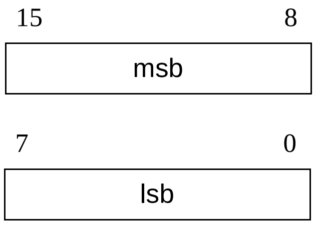
Significand lies in the range $1.0 < \text{significand} < 2.0$, with the integer part implicit.

Zeroes:	
e =	0
f =	0

A.2 Data Representation in the GCOM Protocol

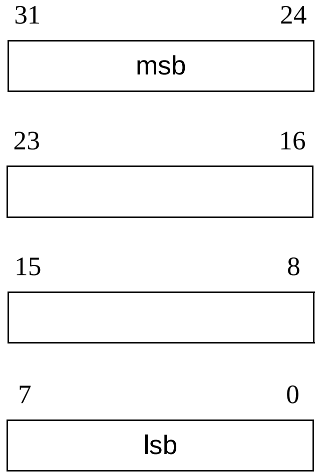
The ABB Master is a 16-bit-oriented system. Therefore, an 8-bit-oriented system must exchange the most significant byte (msb) and the least significant byte (lsb) on a 16-bits basis for data which is defined with a length of 2 bytes (intw) or 4 bytes (intl, real).

Intw (2 bytes) in a message from the ABB Master system is received as follows:



The external computer sends intl in messages to the ABB Master system in the same format.

Intl (4 bytes) in a message from the ABB Master system is received as follows:



The external computer sends intl in messages to the ABB Master system in the same format.

A.3 Definitions

Biased Exponent

> The sum of the exponent and a constant (Bias) chosen to make the biased exponent's range non-negative.

Binary Floating Point Number

> A bit string characterized by three components: a sign, a signed exponent, and a significand. Its numerical value, if any, is the signed product of its significand and two raised to the power of its exponent. A bit string is not always distinguished from a number it may represent.

Exponent

> That component of a binary floating point number which signifies the power to which two is raised in determining the value of the represented number. Occasionally, the exponent is called signed or unbiased exponent.

Fraction

> The field of the significand that lies to the right of its implied binary point.

Normalized

> If the number is nonzero, shift its significand left while decrementing its exponent until the leading significand bit becomes one; the exponent is regarded as if its range were unlimited. If the significand is zero, the number becomes normal zero. Normalizing a number does not change its sign.

Normal zero

The exponent is the format’s minimum and the significand is zero. Normal zero may have either a positive or negative sign. Only the extended format has any unnormalized zeroes.

Significand

That component of a binary floating point number which consists of an explicit or implicit leading bit to the left of its binary point and a fraction field to the right of the binary point.

Appendix B Global Channel Identities

B.1 Gloabal Channel Identities

GCHANNEL CXADSP	1	DSC, receive of DSP
GCHANNEL CXADSD	2	DSC, receive of APT request
GCHANNEL CXACSM	3	Clock sync, synchronization message channel
GCHANNEL DCBCB0	4	Order to AI-object
GCHANNEL DCBCC0	5	Order to AO-object
GCHANNEL DCBCD0	6	Order to DI-object
GCHANNEL DCBCE0	7	Order to DO-object
GCHANNEL DCBCF0	8	
GCHANNEL DCBCG0	9	
GCHANNEL DCBCH0	10	Order to PIDCON-object
GCHANNEL DCCC45	11	
GCHANNEL DCBCI0	12	
GCHANNEL DCBCL0	13	Order to Simp. handler
GCHANNEL DCTC50	14	Order to TTD manip. handler
GCHANNEL DCLC10	15	
GCHANNEL DCCC20	16	Processevent.
GCHANNEL DCBCJ0	17	Order to MANSTN-object
GCHANNEL DCBCK0	18	Order to RATIOS-object
GCHANNEL DCUCE1	19	MMCX-object See also application
GCHANNEL DCUCE2	20	MMCX-object "
GCHANNEL DCUCE3	21	MMCX-object "
GCHANNEL DCUCE4	22	MMCX-object "
GCHANNEL DCDC21	23	Answer to dialog from MMI 1
GCHANNEL DCDC22	24	Answer to dialog from MMI 2
GCHANNEL DCDC23	25	Answer to dialog from MMI 3
GCHANNEL DCPC11	26	For presentation MMI 1
GCHANNEL DCPC12	27	For presentation MMI 2
GCHANNEL DCPC13	28	For presentation MMI 3
GCHANNEL DCSC00	29	Data subscription
GCHANNEL DCCC40	30	Systemevent
GCHANNEL DCCC91	31	
GCHANNEL DCCC92	32	
GCHANNEL DCCC93	33	
GCHANNEL DCCC94	34	
GCHANNEL DCCC95	35	
GCHANNEL DCCC96	36	
GCHANNEL DCCC97	37	
GCHANNEL DCCC98	38	

GCHANNEL DCCC99	39	
GCHANNEL DCLC20	40	
GCHANNEL DCLC30	41	
GCHANNEL DCDC24	42	
GCHANNEL DCPC14	43	For presentation MMI 4
GCHANNEL DCBCM0	44	
GCHANNEL DCBCP0	45	
GCHANNEL DCWC10	46	Event subscription
GCHANNEL DCWC11	47	
GCHANNEL DCWC00	48	
GCHANNEL CXNMC2	49	Symbol name translation request
GCHANNEL CXACND	50	EXCOM, command channel
GCHANNEL CXARP0	51	EXCOM, reply channels
GCHANNEL CXARP1	52	--- " ---
GCHANNEL DCCC9B	53	
GCHANNEL DCCC9C	54	
GCHANNEL DCCC9D	55	
GCHANNEL DCCC9E	56	
GCHANNEL DCCC9F	57	
GCHANNEL DCCC9G	58	
GCHANNEL DCCC9H	59	
GCHANNEL DCCC9I	60	
GCHANNEL DCCC9J	61	
GCHANNEL CXAD00	65	MasterBus, intra process area communication channel
GCHANNEL DCAC01	66	
GCHANNEL DCAC02	67	
GCHANNEL DCAC03	68	
GCHANNEL DCAC04	69	
GCHANNEL DCAC05	70	
GCHANNEL DCAC06	71	
GCHANNEL DCAC07	72	
GCHANNEL DCAC08	73	
GCHANNEL DCAC09	74	
GCHANNEL DCAC10	75	
GCHANNEL CXARP2	76	EXCOM, reply channels
GCHANNEL CXARP3	77	--- " ---
GCHANNEL CXABRQ	78	Clock sync, back up request channel
GCHANNEL CXABRP	79	Clock sync, back up response channel
GCHANNEL CXNMC1	80	Answer to symbolic name translation from MV.
GCHANNEL IBCX00	256	Timesync
GCHANNEL IBCX30	257	Objecttrend
GCHANNEL IBCX41	258	Communication status

GCHANNEL IBCX01	259	Timesync request
GCHANNEL IBCX42	260	Communication status request
GCHANNEL IBCX70	261	Object name resolving
GCHANNEL IBCG41	274	Network status, gateway
GCHANNEL IBCG43	275	Network status, gateway
GCHANNEL IBCG45	276	Network status, gateway
GCHANNEL DCUCE5	268	MMC_X object handler
GCHANNEL DCUCE6	269	MMC_X object handler
GCHANNEL DCUCE7	270	MMC_X object handler
GCHANNEL DCUCE8	271	MMC_X object handler
GCHANNEL DCUCE9	272	MMC_X object handler
GCHANNEL DCUCEA	273	MMC_X object handler

Appendix C Signal Identities

C.1 Signal Identities

```
new_pic             = 1;
sub_req             = 2;
order               = 3;
sel_answer          = 4;
sub_clear           = 5;
ref_change          = 6;
page_req            = 7;
interface_cupdate   = 8;
event_text          = 9;
single_text         = 10;

syst_text           = 11;
displ_gen           = 12;
d_eupdate           = 13;
a_cupdate           = 14;
na_demand           = 15;
la_demand           = 16;
del_and_data        = 17;  (* DCD***-users only *)
pro_event           = 18;
sys_event           = 19;
analys_1            = 20;  (* DCB***-users only *)

mmi_code            = 21;
mmi_ccode           = 22;
mmi_bcode           = 23;
off_aud_al          = 24;
on_aud_al           = 25;  (* DCC***-users only *)
text_handling       = 26;  (* DCC***-users only *)
list_update         = 27;  (* DCC***-users only *)
du_anl_1            = 28;  (* DCB***-users only *)
du_write            = 29;  (* DCB***-users only *)
du_read             = 30;  (* DCB***-users only *)

s_pro_event         = 31;  (* DCB***-users only *)
pid_req             = 32;  (* DC****-users DCXA03, DCPA03 *)
change_presref      = 33;
timeout_aud_al      = 34;  (* DCC**-users only *)
read_time_req       = 35;  (* DCX***-users DCXA03,DCXA04 *)
o2_eupdate          = 36;
o2_cupdate          = 37;
o2_dupdate          = 38;
```

```
o3_eupdate          = 39;
o3_cupdate          = 40;

o3_dupdate          = 41;
seq_eupdate         = 42;
seq_cupdate         = 43;
seq_dupdate         = 44;
send_req            = 45;
int_req             = 46;
reg_cupdate         = 47;
reg_dupdate         = 48;
mmi_pcode           = 49;
mmi_start_up        = 50;

list_picture        = 51;
pid_cupdate         = 52;
pid_n_dupdate       = 53;  (* pid_l_dupdate = 92 , pidDevDupdate = 93 *)
dDevDupdate         = 54;
aiDevDupdate        = 55;
aoDevDupdate        = 56;
o2DevDupdate        = 57;
o3DevDupdate        = 58;
seqDevDupdate       = 59;
data_order          = 60;

data_change         = 61;
data_update         = 62;
data_answer         = 63;
obf_eupdate         = 64;
user01_update       = 65;
user02_update       = 66;
user03_update       = 67;
user04_update       = 68;
pc_seq_sync         = 69;
ttd_request         = 70;

ttd_data            = 71;
ttd_subdata         = 72;
ttd_infdata         = 73;
ttd_answer          = 74;
search_cond         = 75;
pro_status          = 76;
alist_req           = 77;
rs_cupdate          = 78;
rs_n_dupdate        = 79;  (* rs_l_dupdate = 94, rsDevDupdate = 95 *)
ms_cupdate          = 80;
```

```
ms_n_dupdate        = 81;  (* ms_l_dupdate = 96, msDevDupdate = 97 *)
pres_pointer        = 82;
pr_1_line           = 83;
prlinent            = 84;
prlinetab           = 85;
pr_tmout            = 86;
VT_On               = 87;
VT_Off              = 88;
ttd_confdata        = 89;
mmi_clear           = 90;

db_change           = 91;
pid_l_dupdate       = 92;
pidDevDupdate       = 93;
rs_l_dupdate        = 94;
rsDevDupdate        = 95;
ms_l_dupdate        = 96;
msDevDupdate        = 97;
go_cupdate          = 98;
go_n_dupdate        = 99;
go_l_dupdate        = 100;

goDevDupdate        = 101;
rep_aud_al          = 102;
pulse_aud_al        = 103;
stat_node           = 104;   (* dcxswnode *)
stat_deup           = 105;   (* dcxswdeup *)
stat_pmup           = 106;   (* dcxswpmup *)
node_init           = 107;   (* dcxswinit *)
node_resp           = 108;   (* dcxswresp *)
cns_req             = 109;   (* dcxswreq*)
cns_fade            = 110;   (* dcxswfade *)
```

Special signal identities for MasterGate

```
iagw_error              = -1; (* error codes from MasterGate (MG)*)
iacl_sync               = -2; (* MG clock sync. from MG *)
iacl_fr_ec              = -3; (* MG clock sync. from ex. comp. *)
iacl_req_fr_ec          = -4; (* MG clock sync. request *)
iacom_ec                = -5; (* MG communication status *)
iacomreq_ec             = -6; (* MG communication status request *)
iaGW_symbol             = -22; (* MG symbol name translation *)
```

Appendix D Operation Codes and Properties

The operation code defines the order type. The operation property is a closer definition of the order type.

D.1 Operation Codes

Numbers within brackets show relations to operation properties.

```
select_dcx              = 1;        (*1*)
deselect_dcx            = 2;        (*1*)
onopenstart_dcx         = 3;        (*6*)
offclose_dcx            = 4;        (*6*)
stop_dcx                = 5;        (*1*)
acknowledge_dcx         = 6;        (* Event properties *)
set_unack_dcx           = 7;        (* Event properties *)
incr_percent_dcx        = 8;        (*6,7*)
decr_percent_dcx        = 9;        (*6,7*)
small_increase_dcx      = 10;       (*6,7*)

large_increase_dcx      = 11;       (*6,7*)
small_decrease_dcx      = 12;       (*6,7*)
large_decrease_dcx      = 13;       (*6,7*)
value_change_dcx        = 14;       (*3*)
block_dcx               = 15;       (*4*)
deblock_dcx             = 16;       (*4*)
out_of_service_dcx      = 17;       (*1*)
in_service_dcx          = 18;       (*1*)
status_check_dcx        = 19;       (*1*)
set_cond_dcx            = 20;       (*8*)

reset_cond_dcx          = 21;       (*8*)
(* flash_out_dcx        = 22;       ( 1 )   NOT YET IMPLEMENTED *)
(* pos_pulse_out_dcx    = 23;       ( 1 )   NOT YET IMPLEMENTED *)
(* neg_pulse_out_dcx    = 24;       ( 1 )   NOT YET IMPLEMENTED *)
manual_dcx              = 25;       (*1*)
auto_dcx                = 26;       (*1*)
mode_e1_dcx             = 27;       (*1*)
mode_e2_dcx             = 28;       (*1*)
mode_e3_dcx             = 29;       (*1*)
hold_seq_dcx            = 30;       (*1*)

step_seq_dcx            = 31;       (*1*)
uncond_seq_dcx          = 32;       (*1*)
jump_seq_dcx            = 33;       (*1*)
remote_oper_dcx         = 34;       (*1*)
```

```
central_oper_dcx      = 35;       (*1*)
local_oper_dcx        = 36;       (*1*)
spare_1_seq_dcx       = 37;       (*1*)
spare_2_seq_dcx       = 38;       (*1*)
cond_seq_dcx          = 39;       (*1*)
man_forced_dcx        = 40;       (*1*)

AckPosFault_dcx       = 41;       (*1*)
set_dcx               = 42;       (*9*)
reset_dcx             = 43;       (*9*)
RetransSymbols_dcx    = 44;       (*10*)
```

D.2 Operation Properties

Numbers within comment brackets show relations to operation codes.

```
dummy_prop_dcx        = 0;        (*1,5*)
main_dcx              = 1;        (*6*)
spare_dcx             = 2;        (*6*)
setpoint_dcx          = 3;        (*3,7*)
output_dcx            = 4;        (*3,7,4*)
act_value_dcx         = 5;        (*3*)
position_dcx          = 6;        (*5*)
lowlim1_dcx           = 7;        (*3,5*)
lowlim2_dcx           = 8;        (*3,5*)
hilim1_dcx            = 9;        (*3,5*)
hilim2_dcx            = 10;       (*3,5*)

mmc_control_dcx       = 11;       (*4*)
dist_print_dcx        = 12;       (*4*)
dist_alarm_dcx        = 13;       (*4*)
fault_print_dcx       = 14;       (*4*)
fault_alarm_dcx       = 15;       (*4*)
pos_print_dcx         = 16;       (*4*)
pos_alarm_dcx         = 17;       (*4*)
proc_update_dcx       = 18;       (*4*)
fault1_alarm_dcx      = 19;       (*4*)
fault1_print_dcx      = 20;       (*4*)

fault2_alarm_dcx      = 21;       (*4*)
fault2_print_dcx      = 22;       (*4*)
fault3_alarm_dcx      = 23;       (*4*)
fault3_print_dcx      = 24;       (*4*)
fault4_alarm_dcx      = 25;       (*4*)
fault4_print_dcx      = 26;       (*4*)
ratio_dcx             = 27;       (*3*)
bias_dcx              = 28;       (*3*)
ratio_hilim_dcx       = 29;       (*3*)
```

```
ratio_lolim_dcx      = 30;      (*3*)

GeneReal_dcx         = 31;      (*3*)
a1_dcx               = 32;      (*8*)
asp_dcx              = 33;      (*8*)
a0_dcx               = 34;      (*8*)
auto_en_dcx          = 35;      (*8*)
ca1_dcx              = 36;      (*8*)
ca0_dcx              = 37;      (*8*)
cm1a_dcx             = 38;      (*8*)
cm1b_dcx             = 39;      (*8*)
cm0a_dcx             = 40;      (*8*)

cm0b_dcx             = 41;      (*8*)
jump_pos_dcx         = 42;      (*3*)
no_of_turns_dcx      = 43;      (*3*)
interv_time_dcx      = 44;      (*3*)
GeneIntl_dcx         = 45;      (*3*)
Gene01Bool_dcx       = 46;      (*3,9*)
seqindprint_dcx      = 47;      (*4*)
seqindalarm_dcx      = 48;      (*4*)
integration_dcx      = 49;      (*4*)
derivation_dcx       = 50;      (*4*)

gain_dcx             = 51;      (*3*)
ti_dcx               = 52;      (*3*)
td_dcx               = 53;      (*3*)
tf_dcx               = 54;      (*3*)
dead_zone_dcx        = 55;      (*3*)
dev_hilim_dcx        = 56;      (*3*)
dev_lolim_dcx        = 57;      (*3*)
setp_hilim_dcx       = 58;      (*3*)
setp_lolim_dcx       = 59;      (*3*)
outp_hilim_dcx       = 60;      (*3*)

outp_lolim_dcx       = 61;      (*3*)
Gene02Bool_dcx       = 62;      (*3,9*)
Gene03Bool_dcx       = 63;      (*3,9*)
Gene04Bool_dcx       = 64;      (*3,9*)
Gene05Bool_dcx       = 65;      (*3,9*)
Gene06Bool_dcx       = 66;      (*3,9*)
Gene07Bool_dcx       = 67;      (*3,9*)
Gene08Bool_dcx       = 68;      (*3,9*)
Gene09Bool_dcx       = 69;      (*3,9*)
Gene10Bool_dcx       = 70;      (*3,9*)
```

```
Gene11Bool_dcx        = 71;      (*3,9*)
Gene12Bool_dcx        = 72;      (*3,9*)
Gene13Bool_dcx        = 73;      (*3,9*)
Gene14Bool_dcx        = 74;      (*3,9*)
Gene15Bool_dcx        = 75;      (*3,9*)
Gene16Bool_dcx        = 76;      (*3,9*)
Neg_Corr_dcx          = 77;      (*3*)
Pos_Corr_dcx          = 78;      (*3*)
Max_Hilim_dcx         = 79;      (*3*)
MaxLim_dcx            = 80;      (*3*)

MinLim_dcx            = 81;      (*3*)
Dev_Maxlim_dcx        = 82;      (*3*)
Dev_Minlim_dcx        = 83;      (*3*)
alarm_dcx             = 84;      (*4*)
print_dcx             = 85;      (*4*)
GroupAlarm_dcx        = 86;      (*4*)
pgm_alarm_dcx         = 87;      (*4*)
ext_alarm_dcx         = 88;      (*4*)
auto_alarm_dcx        = 89;      (*4*)
GenSPmmc_dcx          = 90;      (*3*)

GenSPImmc_dcx         = 91;      (*3*)
GenOPmmc_dcx          = 92;      (*3*)
GenMVmax_dcx          = 93;      (*3*)
GenMVmin_dcx          = 94;      (*3*)
GenOPmax_dcx          = 95;      (*3*)
GenOPmin_dcx          = 96;      (*3*)
GenSPmax_dcx          = 97;      (*3*)
GenSPmin_dcx          = 98;      (*3*)
GenmaxINTL_dcx        = 99;      (*3*)
GenminINTL_dcx        = 100;     (*3*)

GenEvent_dcx          = 101;     (*4*)
CalcValue_dcx         = 102;     (*3*)
All_dcx               = 103;     (*10*)
OneNode_dcx           = 104;     (*10*)
```

INDEX

A
a_cupdate 3-56
access type 3-20
Acknowledge and sequence handling 1-12
Address handling 1-8
Address Part 3-15
aiDevDupdate 3-59
Answer on Data Subscription 3-24
Answer on data subscription 3-56
aoDevDupdate 3-58
Application Functions 3-1
Application layer 3-14

B
Boolean A-2

C
Capacity 3-2
Character A-2
Clock Synchronization 3-49
Communication Status 3-50, 3-99

D
d_eupdate 3-62
Data Part Object Functions 3-18
Data Part System Functions 3-46
Data Representation A-1
Data Subscription 3-20, 3-55
Data transfer handling 1-10
dDevDupdate 3-62
Destination address 3-16, 3-17
DSC 3-45
DSC cyclic 3-45
DSC request 3-46

E
Enumerated scalar A-3
Event Handling 3-52, 3-100

G
GCOM/IEEE 802.3 format 1-7
Global Channel Identities B-1

H
Historical Time Tagged Data 3-26, 3-88

I
I am here”- message 1-9
illegal values 3-29
Integer A-1

L
la_demand 3-57

M
Message information 3-16, 3-18
MG-event subscription 3-52
mmi number 3-19
mmi_clear 3-23, 3-24, 3-55
ms_cupdate 3-73
ms_l_dupdate 3-75
ms_n_dupdate 3-74
multicast address 1-9

N
na_demand 3-56

O
Object Functions 3-1
Operation Codes D-1
Operation Properties D-2
Order 3-93

P
Parameters for MANSTN 3-81
Parameters for PIDCON 3-76
Parameters for RATIOSTN 3-83
Performance 3-2
pid_cupdate 3-66
pid_l_dupdate 3-68
pid_n_dupdate 3-67
Pointer A-3

R
Real A-4
reference type 3-18
rs_cupdate 3-70
rs_l_dupdate 3-72
rs_n_dupdate 3-71

S
sel_answer 3-94

Set A-2
Signal Identities C-1
Signal identity 3-17, 3-18
Source address 3-16, 3-17
stat_deup 3-51, 3-99
Status for AI and AO objects 3-60
Status for DI and DO object 3-63
Status for MANSTN 3-82
Status for PIDCON 3-79
Status for RATIOSTN 3-85
String A-3
sub_clear 3-23, 3-55
sub_req 3-23, 3-50, 3-55, 3-99
Symbolic Name Translation 3-47
sys_event 3-53, 3-100
syst_text 3-94
System Event 3-53
System Functions 3-1
System Text 3-94

T

Transit-handling 1-9
TTD 3-26, 3-88
TTD Data Answer 3-29
TTD Data Request 3-27
TTD Information Answer 3-35
TTD Information Request 3-34
ttd_data 3-29, 3-90
ttd_infdata 3-35, 3-91
ttd_request 3-27, 3-34, 3-88
Tutorial 3-3

3BSE 000 165R0001

February, 1997